Knowledge Representation for Chatbot Design
preliminary report

Enrico Franconi[0000−0002−7814−515X] and Sergio Tessaris[0000−0002−3156−2669]
Free University of Bozen–Bolzano, Italy

Abstract. Decision trees are useful structures to design Chatbots, however their design and maintenance are often done by hand. This introduces a level of complexity which can be difficult to manage by the domain experts, as well as suffers of design biases that might limit the interaction of the Chatbot with the user. In this work we introduce general formal framework to characterise the definition and construction of Decision Trees using semantic technologies; moreover we show an implementation of this framework which combines automated reasoning and machine learning techniques.

Keywords: Chatbot design · Decision Trees · Knowledge Representation · Automated Reasoning

1 Introduction

A chatbot is a software application that conducts a conversation via textual or text-to-speech methods [17]. The term has been introduced in the mid 90s in the context of the Loebner Prize competition [12]. Nowadays chatbots are widely used in industry and its market share is expected to grow further [2]. On the market there are several chatbot frameworks (e.g. Microsoft Bot Framework\(^1\), Google DialogFlow\(^2\), IBM Watson\(^3\), Amazon Lex\(^4\)) providing services for natural language processing, classification of user answers, and user interface management; moreover most of them also give the possibility of automatic deployment of the Bots. An interesting discussion of the different frameworks and approaches can be found in [7].

Among the diverse applications of this technology [17] in this paper we use as a sample scenario in the domain of legal advising. In our scenario the chatbot is a first contact with a potential client and, according to her needs specified during the dialogue, the client request is eventually categorised and referred to one of the solicitors of the firm. The task of the chatbot is to collect all the relevant information that would help the solicitor to understand the situation of the potential client and provide the right advice. As a matter of fact, the legal domain is rather intricate, as it requires to be captured in a quite fine grained and precise way, with options which may interrelate positively or negatively via convoluted knowledge.

\(^1\) https://dev.botframework.com
\(^2\) https://dialogflow.com
\(^3\) https://www.ibm.com/cloud/watson-assistant
\(^4\) https://aws.amazon.com/lex
The legal advising domain is an excellent testbed for flow-based chatbot deployment because the kind of requests the firm receives should be categorised (intent selection) and the interaction with the client follows a rather complex but well defined “script” where the relevant information is collected by the solicitor. In this kind of chatbot the conversation works towards pre-defined goals. The domain should be accurately modelled and captured by Bot, so to not allow for any divergence from the intended conversation options.

The developer should design the logic flow of how these services should composed and executed, that is the “script” of the Bot. Usually this is done via a proprietary language or ECMAScript programs. One of the way to conceptualise this script is via the notion of acyclic Decision Graphs (or, without loss of generality, Decision Trees - DT)[11].

In response to a conversation with a user, a chatbot system uses a decision tree model by dynamically traversing the model, and based on the input and properties of the input data, it identifies which leaf of the decision tree should be employed. In some scenarios, the leaf may be resolved using pre-defined properties and/or dynamic functions.

A Decision Tree as a workflow for information gathering. The leaves of the tree constitute a decision with an associated “payload” of data collected during the acquisition process. At each node in the decision tree the Bot asks a multiple choice question and proceeds to the next node according to the given answer. Figure 1 shows a DT\(^5\) and an example of user interaction based on it.

---

\(^5\) Example taken from [13].

---

In the literature, flow based techniques for chatbots are referred (see [4]) as conversation flow diagrams [6], decision trees [11], dialogue graphs [8], dialogue trees [7], stories [18], paths [10], and flow of intents [15].

In most of the cases these decision trees are designed and maintained by hand; which is fine when their size is small, but became quickly unmanageable with complex domains. The number of alternative paths in the flow grows exponentially with respect to

---

Fig. 1. Example of DT and user interaction
the number of variables and the size of their admitted domains. A raw and explicitly maintained decision tree would appear impenetrably complex in any user interface.

Another major general disadvantage of flow-based chatbots is their being boring and potentially following an unnatural conversation path, since they tend to be unaware of the history of the conversation, by not understanding the correlation between already provided information and the the current state.

Fixing the above issues may lead to design bias – in the attempt to limit the complexity and increase the linearity of the conversation– which tend to restrict the scope of the decision tree and to exclude potential useful interaction patterns. Our project partners from a legal firm found themselves in this situation, so we started our collaboration to understand how the design process of Decision Trees could be improved.

Following the analysis and the suggestions from [4], the focus of our work is to support and improve the design process and maintenance of Decision Trees for chatbots. We don’t consider the problem of understanding and classifying the user responses.\(^6\)

We propose a formal declarative specification of the properties of the domain and the decisions, based on well funded Knowledge Representation (KR) technologies, with the support of well known knowledge engineering methodologies. From this declarative specification the implicitly defined, complete, and optimal decision tree can be constructed using algorithms exploiting automated reasoning. The result of the process is a static decision tree which can be deployed on one of the available engines.

The advantage of this schema is a great flexibility of deployment and the efficiency of the chatbot, since the tree generation is performed offline, and the DT design is based on the semantic modelling of the domain. The chatbot designer does not have to manage a decision tree, but she has to carefully model the domain knowledge. The decision tree will be automatically generated out of this design.

The obtained decision tree is optimal both with respect to the size – i.e., each path is always minimal given the already acquired information – and with respect to the semantic interrelations between pieces of information. This means that at any moment of the flow the bot never tries to acquire information implied by the background domain knowledge representation and by the already given information. These two aspects tame the issue of having unnatural conversation paths.

In this paper we introduce a general formal logic-based framework to characterise the definition and construction of optimal Decision Trees using semantic technologies, and we show an implementation of this framework using a combination of symbolic and machine learning techniques.

2 KR for chatbot design

Our work focuses on formally characterising and designing decision trees to guide the user interaction of flow-based chatbots. Each node of the tree is a choice point which requires interaction with the user who provides an answer to a precise question formulated by the chatbot. Once a leaf is reached the system terminates the dialog and provides the outcome of the interaction.

\(^6\) Tools for this task are available in most of the Chatbot frameworks.
Domain experts provide a list of facets that influence the decisions of the Bot (e.g. the Outlook, Humidity, and Wind of Figure 1), together with finite sets\(^7\) enumerating their possible values (e.g. for Outlook they are \{Sunny, Overcast, Rain\}). In addition, they must describe the outcome of the decisions in terms of those features; e.g. if the Outlook is Sunny and the Humidity is High then the answer is No (i.e. do not play). Optionally, experts can include additional informations on the domain; e.g. Humidity being High and Wind being Strong are not compatible.

With our framework these informations are sufficient for building a decision tree which minimises the questions to be asked to the user to reach one of the decisions. In order to achieve this we use a formal language providing support for automated reasoning. The complete description of the framework is outside the scope of this paper and we'll just introduce the main ideas. To represent the knowledge about the domain we use (decidable) fragments of First Order Logic with equality \([5]\), an expressive and widely used language to represent domain knowledge. The choice of restricting to decidable fragments\(^8\) enables the automated construction of the decision trees.

The facets are represented by a (finite) set of names (denoted by \(F\)), and to each facet \(f \in F\) we associate a finite domain \(f_D\). To indicate the fact that a facet assumes a specific value from the domain we use predicate notation; e.g. \(\text{Humidity}(\text{High})\) represents the information that the humidity value is high. Predicates representing facets are used to describe both the additional domain knowledge and the decisions. The former takes the form of a set of formulae (denoted by \(T\)) that must be satisfied, e.g. the knowledge that there cannot be a strong wind and high humidity at the same time could be expressed by the formula \(\neg(\text{Humidity}(\text{High}) \land \text{Wind}(\text{Strong}))\). We consider a finite set of decisions \(D\), and to each \(d \in D\) an associated a set of boolean (i.e. just true or false) closed formulae (which we call queries) \(d_Q\). Queries represent the conditions that enable the Bot to identify a significant situation. E.g. in the example of Figure 1 the decision are Yes and No, and the queries associated to No are \(\text{Outlook}(\text{Sunny}) \land \text{Humidity}(\text{High})\) and \(\text{Outlook}(\text{Rain}) \land \text{Wind}(\text{Strong})\). Note that we do not impose any restriction on the disjointness of the queries; that is more than one query – namely, decision – can be satisfied at the same time. Neither we assume any notion of coverage of the queries – i.e. all the decisions should be satisfiable. Specific applications of this framework may require those or additional restrictions.

To tie the above notions to the definition of Decision Trees we make use of the notion of contexts which are (partial) assignments of the facets to values in their domain. Note that a context is not necessarily a model (in its formal logic meaning) for the background theory because it’s not required to have a value for all facts and in the language there might be additional symbols. A context represents the known information about a given situation. E.g. \{Outlook(\text{Sunny}), \text{Humidity}(\text{Normal}), \text{Wind}(\text{Strong})\} is a context. We are not interested in all the contexts but only those that are “compatible” with both the background theory and the set of decisions. In fact, just a subset of those contexts are significant for relevant classifications; bear in mind that the goal

\(^7\) Note that supporting infinite domains (e.g. unbound numbers) requires more complex techniques which are out of the scope of this work.

\(^8\) I.e. it’s always possible to verify whether a fact, or formula, can be derived from a set of given assertions.
is to guide the user to provide the relevant information, not to classify any arbitrary context. These two properties can be captured by means of satisfiability and logical implication. Firstly we consider the logic encoding of a context $c$ (denoted as $\tilde{c}$) as the formula $\tilde{c} = \bigwedge_{f \in \text{Fac}(f)} f(v)$. E.g. the above context would be $\text{Outlook}(\text{Sunny}) \land \text{Humidity}(\text{Normal}) \land \text{Wind}(\text{Strong})$.

Then those properties are: 1. $\mathcal{T} \cup \tilde{c}$ is satisfiable, and 2. there is $d \in \mathcal{D}$ s.t. $\mathcal{T} \cup \tilde{c} \models \bigvee_{q \in \mathcal{D}_q} q$. In addition we want to minimise redundancy: if $\mathcal{T} \cup \tilde{c} \not\models \bigvee_{q \in \mathcal{D}_q} q$ then for any $c' \subseteq c$ we know that $\mathcal{T} \cup \tilde{c'} \models \bigvee_{q \in \mathcal{D}_q} q$, i.e., $\{d \mid \mathcal{T} \cup \tilde{c} \models \bigvee_{q \in \mathcal{D}_q} q\} \subseteq \{d \mid \mathcal{T} \cup \tilde{c'} \models \bigvee_{q \in \mathcal{D}_q} q\}$. Therefore $c'$ doesn’t add additional useful information. Therefore we require that 3. there is no $c' \subseteq c$ s.t. $\{d \mid \mathcal{T} \cup \tilde{c} \models \bigvee_{q \in \mathcal{D}_q} q\} \subseteq \{d \mid \mathcal{T} \cup \tilde{c'} \models \bigvee_{q \in \mathcal{D}_q} q\}$. We also want to consider only “specific” contexts, that is we avoid contexts which can be extended to identify additional decisions. This can be formalised by the restriction 4. there is no $c' \supseteq c$ ($c'$ compatible with $\mathcal{T}$) s.t. $\{d \mid \mathcal{T} \cup \tilde{c} \models \bigvee_{q \in \mathcal{D}_q} q\} \subset \{d \mid \mathcal{T} \cup \tilde{c'} \models \bigvee_{q \in \mathcal{D}_q} q\}$.

The set of all contexts satisfying the above properties, the so called valid contexts (denoted as $C_{TD}$), identifies those that carry information about at least one decision. To keep the notation simple we’ll use the same symbol to denote also the function mapping each valid context to the set of decisions that they satisfy; i.e.: $C_{TD}(c) = \{d \in \mathcal{D} \mid \mathcal{T} \cup \tilde{c} \models d\}$.

Valid contexts and their mapping to decisions enable us to bridge to the notion of decision trees. In fact, leaves of a tree can be univocally identified by the path connecting them to the root. In a DT each edge is identified by a pair facet/value (i.e. $\bigcup_{f \in \mathcal{F}} \{f\} \times f_D$), therefore paths can be read as contexts. E.g. the DT in Figure 1 the leaves are:

- [(outlook, overcast)]
- [(outlook, sunny), (humidity, high)]
- [(outlook, sunny), (humidity, normal)]
- [(outlook, sunny), (wind, strong)]
- [(outlook, sunny), (wind, weak)]

We can use this correspondence to identify DTs that represent the information of the $C_{TD}$ function [14]. Obviously there are several decision trees representing the same discrete function, therefore we should identify one of those on the basis of its fitness for representing the information acquisition workflow. We identify the following relevant properties: 1. for each leaf $\ell'$ in the tree $\mathcal{T} \cup \ell'$ is satisfiable, and there exists $c \in C_{TD}$ s.t. $c \subseteq \ell'$; 2. for each context $\in C_{TD}$ there is a leaf $\ell'$ in the tree s.t. $c \subseteq \ell'$.

In general our goal is to minimise the interaction with the user and identify an applicable decision with a minimum number of questions. Therefore we focus on “compact” trees with a low average depth. In the next section we demonstrate a specific implementation of this framework, built in the context of a project with a law firm.

### 3 An Example Implementation

The problem of constructing “compact” decision trees from tables containing examples is a well studied problem in the Machine Learning area [13], and there are several techniques and off-the-shelf implementations [11]. The details of such techniques are outside

---

9 We assume a monotonic logic $\mathcal{L}$, relaxing this assumption changes the scenario.

10 The set equality is required because there might be several choices satisfied in a context.
the scope of this paper and the relevant fact is that they all take as input a training table sampling the discrete function and generate a DT that approximates the original function. By constructing the training table corresponding to the $C_{T,D}$ above we can exploit any one of these systems to generate a decision tree. Aside, generating the intermediate $C_{T,D}$ has also the advantage that domain experts can be easily spot modelling problems because a table is much easier to understand than a logic specification.

If the problem of checking satisfiability in the underlying logic $\mathcal{L}$ is decidable, our formulation in Section 2 provides a complete and terminating algorithm for the construction of such a table.\footnote{That’s because the set of all contexts is finite when we restrict to finite domains.} However, such a naive algorithm is bound to have scalability problems because of the optimisation of the potentially huge $C_{T,D}$, and the fact that DT optimisation is a hard problem per se [19]. For the time being we focused on the first problem by considering specific restrictions on the $\mathcal{L}$ language that would enable the use of efficient techniques for the materialisation of $C_{T,D}$.

By analysing the requirements of our project partners we decided to restrict to a language language including database like constraints and simple class hierarchies. This language can be embedded into Answer Set Programming formalism [3], which provides robust and flexible techniques for the declarative specification of optimisation problems, as well as a variety of industrial strength solvers. In particular we used the Clingo ASP system [9].

The details of the encoding are outside the scope of this paper, but in essence we encoded the theory and decision queries by means of ASP constraints and rules respectively. The selection of facts value assignments is encoded by using the so called choice rules. On top of that we instruct the solver to generate all the optimal models according to the properties described in the previous section. From each model we can then extract both the fact value assignments and the decisions by looking at which queries are satisfied. The training table for the DT generation algorithm is derived from all these models.

Although the outcome of a Machine Learning DT construction algorithm from examples is exactly a tree which can be used for information gathering, there are some subtle differences on how we use it w.r.t. the standard classification problem that require same adaptation. All ML algorithms are geared towards the generalisation from a set of examples that are just a sample of the whole function. The consequences is that they are carefully crafted in order to avoid overfitting (i.e. being too specialised w.r.t. the given examples), and they must be able to classify also unseen examples (i.e. assigning decision also to contexts that are not among the training set). In our case both these consequences make the resulting tree unsuitable for our purpose because the generalisation process might introduce errors in assigning decisions to a context, and they might allow contexts that should not be considered (e.g. when they are incompatible with the theory, or do not correspond to any decision).

Fortunately, most of the ML algorithms for DTs can be tuned and adapted to avoid the above problems. We selected the open source Weka library [1] (implementing a variant of the well known C4.5 algorithm [16]) and tuned for our purposes. In detail, we
set the parameters in order to maximise overfitting,\textsuperscript{12} and we pruned the resulting tree to remove branches without examples.

The deployment of our technique in the context of our project partners provided us encouraging results. The firm reckoned that a declarative approach to the tree design has several advantages both in the maintainability of the artefact and on providing a better understanding of the domain.

\section{Discussion}

In this paper we outlined a semantic based framework for the declarative definition of decision trees to implement Chatbots. We also show a specific implementation of this framework using an hybrid technique combining computational logic and machine learning techniques.

Although logic rules have been used to drive chatbots, to the best of our knowledge this is the first work that proposes a \textit{well founded} semantic based approach to the \textit{declarative} definition of (static) \textit{optimal} decision trees that can be deployed on of-the-shelf bot engines.

Our framework is general enough to be adapted to a variety of declarative languages and decision tree restrictions; e.g. single or many valued decisions, binary or higher degree, etc. The formal declarative nature of the framework enables the exploitation of automated reasoning techniques to characterise properties of the implicit tree description. Moreover, we don’t dictate any fixed technique to generate the actual tree from the description which could be tuned according to the specific deployment of the framework.

The presented implementation, although promising, has some limitations mainly related to the scalability of the approach. It is well know that DT optimisation is a difficult problem \cite{19}, so when the tables grow in size the second stage of the construction might became virtually unfeasible. There are ways to control the size of the tables (i.e. the models of the ASP program) but they require a deep understanding of the logic representation of the domain; which is not always available to the domain expert. One solution we have considered is to drop the completeness of the table and to sample the $C_{T,D}$: although this technique is altering the formal properties of our framework it provides a quick and valuable approximated feedback to the domain expert.

There are several directions in which we are applying and extending the technique for the implementation of our framework. On one side we are investigating the performances of different DT construction algorithms in our setting, but we are also considering alternative techniques that would not require the materialisation of the decision table.

Another area in which we are extending our work is the support for infinite domains (e.g. numbers). Our current implementation handles such domains – like free text (names, ids) or numeric values – however this is done by an \textit{a-priori} discretisation of the domain (e.g. text present, or value falling into a specific range). We are investigating techniques to automatically infer such discretisation by looking at the theory and queries.

\textsuperscript{12} Allowing the presence of leaves accounting for a single example.
References


