Alexa, How Can I Reason with Prolog?

Falco Nogatz
University of Würzburg, Department of Computer Science, Germany
falco.nogatz@uni-wuerzburg.de

Julia Kübert
University of Würzburg, Department of Computer Science, Germany
julia.kuebert@stud-mail.uni-wuerzburg.de

Dietmar Seipel
University of Würzburg, Department of Computer Science, Germany
dietmar.seipel@uni-wuerzburg.de

Salvador Abreu
LISP and Department of Computer Science, University of Évora, Portugal
spa@uevora.pt

Abstract

As with Amazon's Echo and its conversational agent Alexa, smart voice-controlled devices become ever more present in daily life, and many different applications can be integrated into this platform. In this paper, we present a framework that eases the development of *skills* in Prolog. As Prolog has a long history in natural language processing, we may integrate well-established techniques, such as reasoning about knowledge with Attempto Controlled English, instead of depending on example phrases and pre-defined slots.

2012 ACM Subject Classification Human-centered computing → Natural language interfaces

Keywords and phrases Prolog, Attempto Controlled English, Voice-Controlled Agents, Controlled Natural Language

Digital Object Identifier 10.4230/OASIcs.SLATE.2019.17

Supplement Material The source codes of *library(race)* and the framework for Alexa skills in Prolog are available on GitHub at `https://github.com/fnogatz/race` and `https://github.com/fnogatz/alexa.pl` (MIT License).

1 Introduction

With Amazon's Echo, voice-controlled devices became available to a large number of customers and developers. As of today, Amazon's voice-controlled intelligent personal assistant service *Alexa* is one of the most used speech-based natural user interfaces [9]. Its popularity is also due to the system's openness to developers: with *Amazon Developer*[1], it is possible to create speech-based applications even without any prior knowledge of natural language processing (NLP); new so-called *skills* can be defined just by providing example phrases. The user's speech-based request is then sent as serialised text to the developer's server. This way, the developer only needs to handle and produce text messages instead of speech, i.e. the text that is returned by the developer's server is synthesised into speech by Amazon's services and spoken back to the user.

Although this division in responsibilities – Amazon converts the speech to text and vice-versa, the developer's web-service receives and returns only text-based messages – eases the development process and lowers the bar for developers who are not familiar with NLP,

[1] Amazon Developer Platform – `https://developer.amazon.com/alexa/`

the resulting skills are limited to the pre-defined example phrases and capable of handling only small placeholders, the so-called *slots*. As a result, there are in fact more than 50,000 skills available[2], but their common types of interactions are only very simple tasks [8], for instance: *check the weather*, *find facts*, *set timers*, or *tell a joke*.

With a controlled natural language (CNL), it is possible to use a more expressive language, which is nevertheless defined to remain easy to parse. The CNL can be integrated with Amazon's Alexa skills by defining an intent that contains only a single slot, whose content is then entirely handled by the server. In this way, the resulting skill is capable of handling much more complicated requests.

Building on the vast experience in applying Prolog to natural language processing, we present its integration with Amazon's Alexa architecture. Prolog has good support for the CNL *Attempto Controlled English* (ACE) [2]. Its reasoner [1] allows one to extend Alexa with a skill to store knowledge, retrieve it, and even infer new knowledge. It turns out this particular skill exposes some of the current limitations when using Amazon's voice service with long, free speech in a single slot.

The remainder of the paper is organised as follows. In Section 2, we give a short introduction into the definition of a skill for Amazon's Alexa service. The basic concepts of ACE are presented in Section 3. In Section 4, we present our proposed architecture for a Prolog web-server that processes ACE via Alexa and finally, the paper ends with concluding remarks in Section 5.

2 Intents for Alexa Skills

Amazon's speech-based applications can be used by various smarthome devices. Besides the smart speakers of the *Echo* family, there are integrations for all modern mobile platforms as well as smart TVs. Their common basis are *skills* – applications that can be created and released by developers. Its usage requires an internet connection, since the recorded user's speech is sent to Amazon's servers. With the help of the *Alexa Voice Service*[3], it is converted into text, and then sent to the developer's server.

$$\underbrace{\textit{Alexa,}}_{1}\ \underbrace{\textit{ask}}_{2}\ \underbrace{\textit{Prolog}}_{3}\ \underbrace{\textit{to remember that}}_{4}\ \underbrace{\textit{Sam is a human}}_{5}$$

Figure 1 Structure of an example intent.

A single skill provides multiple actions, the *intents*. Besides the ones defined by the developer (e.g., "tell a joke"), Amazon provides built-in intents, e.g., to ask for help, or cancel the current action. The general structure of an intent formulated by a user is given in the example of Figure 1. It consists of the following five parts:

1. *Wake word.* By speaking this word, the smarthome device is activated. It can be changed by the user to "Alexa", "Echo", or "Computer".
2. *Phrase to start a skill.* To start or use a skill, the user can use one of the phrases as defined by Amazon, e.g., "ask", "start", or "open".

[2] IFA 2018: Alexa Devices Continue Expansion Into New Categories and Use Cases, Amazon's Developer Blog, Sep 1, 2018 – `https://developer.amazon.com/blogs/alexa/post/85354e2f-2007-41c6-b946-5a73784bc5f3/`

[3] Alexa Voice Service – `https://developer.amazon.com/alexa-voice-service`

3. The *invocation name* is a name defined by the developer that identifies the used skill. In our example application, we use “Prolog”, which could be easily changed.
4. *Intent name.* Since a skill can provide multiple actions, the user can specify the requested intent. In the example of Figure 1, we use “remember” to store knowledge in the knowledge base.
5. An intent can optionally process any number of *slot values.* These are placeholders with pre-defined data types, e.g. `AMAZON.Number`, that can be extended by self-defined slot types.

For our integration of Alexa with ACE, we use the invocation name “Prolog”, and define three intents, each with a single slot: (i) “remember” takes a sentence, tests for its consistency with the prior knowledge, and stores it as part of the knowledge base; (ii) “prove” checks if the given sentence can already be inferred from the current knowledge base; and (iii) “question” can be used to ask for an answer. Based on these intents, additional actions could be defined, for instance to forget parts of the stored knowledge.

3 ACE: Syntax, Interpretation, and Reasoning

In most existing Alexa skills, the slots are expected to contain just a single word, often as part of a list of pre-defined values (e.g., an animal),[4] or free text of only a few words (e.g., “tell a joke about [topic]”, where the topic is generally described in up to three words). In our application, the slots are expected to contain several words, like “Sam is a human”, as seen in Figure 1 for the intent “remember”. In order to process the sentence or question given in a slot, we make use of Attempto Controlled English.

3.1 Attempto Parsing Engine

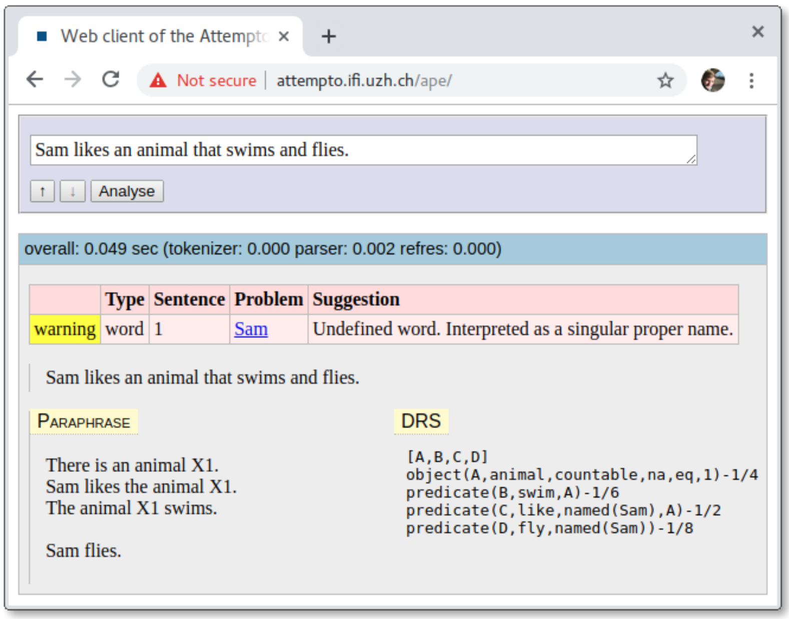
The language definition of ACE consists of three parts. The *vocabulary* contains pre-defined function words (e.g., determiners, conjunctions, and query words), a basic lexicon with content words (e.g., proper names, verbs, and adjectives), and typical fixed phrases to state knowledge or ask questions about (e.g., “there is”, and “it is false that”). Their combination into valid ACE sentences is defined by the *construction rules.* To avoid ambiguities, ACE defines several *interpretation rules* that specify the conceivable meaning of, e.g., relative clauses, and prepositional phrases.

In ACE, sentences always relate to the previous. To guarantee a distinct internal representation, ACE makes use of the following strategies:

- *Restricted language.* ACE consists of a pre-defined set of known words. Unknown words can be prefixed by their word-class, for instance “p:Alexa” for the proper name “Alexa”. Because this is only an option for written but not spoken text, we instead make use of ACE’s best-effort guess for the correct word-class.
  ACE also requires that every noun is used together with a noun marker or quantifier. That means the indefinite article “a” in “Sam likes an animal” stands for the existential quantification and is strictly needed; it is equivalent to “There is an animal that Sam likes”.

  Besides all restrictions, ACE’s language is not limited to just simple, declarative sentences – determined by a subject, verb, and object. Composite sentences that combine declarative sentences by constructors like “or”, “not”, and “if” are also allowed.

[4] Amazon provides several slot types for commonly used categories, e.g. `AMAZON.Animal`. These slot types increase the chance to correctly identify the spoken word.


Web client of the Attempto
Not secure | attempto.ifi.uzh.ch/ape/
↑ ↓ Analyse

Sam likes an animal that swims and flies.

overall: 0.049 sec (tokenizer: 0.000 parser: 0.002 refres: 0.000)

	Type	Sentence	Problem	Suggestion
warning	word	1	Sam	Undefined word. Interpreted as a singular proper name.

Sam likes an animal that swims and flies.

PARAPHRASE

There is an animal X1.
Sam likes the animal X1.
The animal X1 swims.

Sam flies.

DRS

```
[A,B,C,D]
object(A,animal,countable,na,eq,1)-1/4
predicate(B,swim,A)-1/6
predicate(C,like,named(Sam),A)-1/2
predicate(D,fly,named(Sam))-1/8
```

Figure 2 Screenshot of the APE web client.

- *Solving of ambiguities with constraints and interpretation rules.* The easiest way to avoid ambiguities is to constrain the language to not allow ambiguous constructs, when unambiguous alternatives are available. For instance, the sentence “Sam likes an animal that swims and flies” will have the unequivocal meaning that Sam flies. To express that the animal is able to swim and fly, ACE requires the repetition of the relative pronoun “that”, so the sentence should be “Sam likes an animal that swims and that flies” instead.

  However, not all ambiguities can be safely removed from ACE without making the sentences stilted and artificial. Therefore, ACE specifies interpretation rules that define the scope of sub-sentences, quantifiers, etc.

- *Paraphrasing.* The last step in recognising an erroneous interpretation of the given sentence is to return a paraphrase to the user. For instance, the (unintended) paraphrase of the sentence “Sam likes an animal that swims and flies” is given in Figure 2.

The three parts – vocabulary, construction rules, and interpretation rules – are combined in the *Attempto Parsing Engine* (APE) [3]. It translates a given ACE text into *discourse representation structure* (DRS) [7], which uses a variant of the language of first-order logic. Since APE is implemented in Prolog, the DRS representation can be also modelled as a Prolog term, in which content words are represented by atoms and relationships by Prolog predicates with shared variables.

For a more detailed introduction into ACE, we refer to [4, 6]. A short overview about the language features supported by ACE is given in the online documentation at `http://attempto.ifi.uzh.ch/site/docs/ace_nutshell.html`. There is a public APE web service available at `http://attempto.ifi.uzh.ch/ape/`. In Figure 2, we present a screenshot of its web client for the example sentence “Sam likes an animal that swims and flies”, with its corresponding paraphrase and DRS, showing the interpretation assumed for ACE.

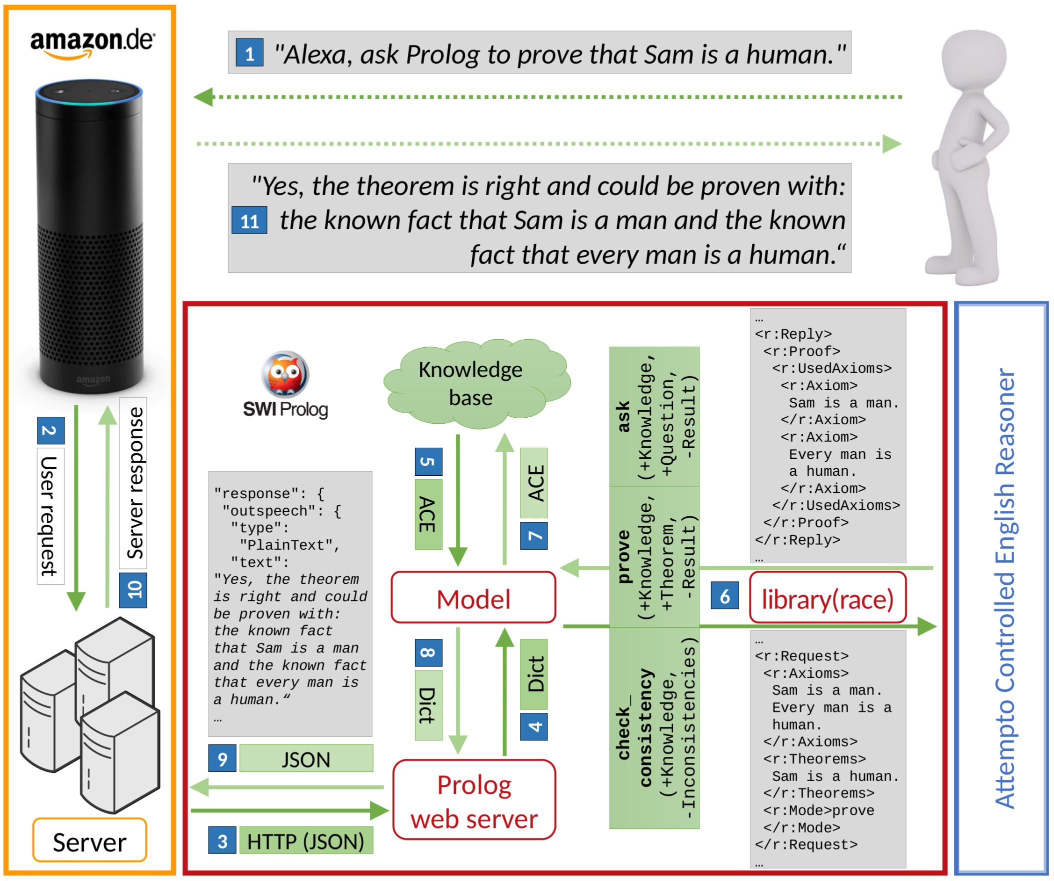
3.2 The ACE Reasoner

Because DRS is a variant of first-order logic, it can be used to formulate theorems. The knowledge expressed as ACE sentences can be combined to infer new knowledge, to prove theses or to answer questions. In [1], Fuchs presents first-order reasoning for ACE texts. Its implementation is called *ACE Reasoner* (or *RACE* for short), and is based on Prolog. Like for ACE, there is a public web service available at `http://attempto.ifi.uzh.ch/race/`. Its web client supports three operations to (i) check the consistency of a given ACE text, (ii) to prove a theorem based on known axioms, and (iii) to answer a query based on known axioms.

In addition to the web client, RACE can also be accessed using a SOAP interface.[5] Because the source code of RACE is not freely available, this XML-based request-reply protocol is the only way to use RACE in connection with our Alexa skill.

4 Service Architecture

In Figure 3, we describe the service architecture for our integration of ACE into an Alexa skill using Prolog. In this example, the user wants to prove that Sam is a human, based only on previously expressed knowledge.



amazon.de
1 "Alexa, ask Prolog to prove that Sam is a human."
11 "Yes, the theorem is right and could be proven with: the known fact that Sam is a man and the known fact that every man is a human."
2 User request
10 Server response
Server
SWI Prolog
Knowledge base
5 ACE
ACE 7
Model
8 Dict
Dict 4
Prolog web server
9 JSON
3 HTTP (JSON)
ask (+Knowledge, +Question, -Result)
prove (+Knowledge, +Theorem, -Result)
check_consistency (+Knowledge, -Inconsistencies)
6 library(race)
Attempto Controlled English Reasoner

```
"response": {
 "outspeech": {
  "type":
   "PlainText",
  "text":
"Yes, the theorem
is right and could
be proven with:
the known fact
that Sam is a man
and the known fact
that every man is
a human.“
…
```

```
…
<r:Reply>
 <r:Proof>
  <r:UsedAxioms>
   <r:Axiom>
    Sam is a man.
   </r:Axiom>
   <r:Axiom>
    Every man is
    a human.
   </r:Axiom>
  </r:UsedAxioms>
 </r:Proof>
</r:Reply>
…
```

```
…
<r:Request>
 <r:Axioms>
  Sam is a man.
  Every man is a
  human.
 </r:Axioms>
 <r:Theorems>
  Sam is a human.
 </r:Theorems>
 <r:Mode>prove
 </r:Mode>
</r:Request>
…
```

Figure 3 Service Architecture for the integration of RACE in an Alexa skill.

[5] RACE Webservice – `http://attempto.ifi.uzh.ch/site/docs/race_webservice.html`

4.1 Speech-to-text by Alexa Voice Service

The smarthome device sends the spoken user request to Amazon's *Alexa Voice Service* (AVS), where it is translated into a textual representation. As part of this process, the requested skill is recognised, as well as the needed intent, and possibly the values for the slots of this particular intent.

The development of a new skill requires an Amazon Developer account. The skill creation consists of several steps, but first, the name of the application and supported languages have to be chosen. Amazon provides several pre-defined *models*, e.g., for a typical request to a media player. By defining a *custom skill*, the model can be adapted to fit our three intents "remember", "prove" and "question". In particular, they are created as example phrases in the so-called *interaction model*, each with a single slot. They are of the data type `AMAZON.SearchQuery`, which does not restrict the slot value, but consequently often comes with inaccuracies.

Listing 1 Definition of the Prolog interaction model as JSON.

```
{ "interactionModel": {
    "languageModel": {
      "invocationName": "prolog",
      "intents": [
        { "name": "remember", ... },
        { "name": "question", ... },
        { "name": "prove",
          "slots": [
            { "name": "proveslot",
              "type": "AMAZON.SearchQuery" } ],
              "samples": [ "prove {proveslot}" ], ... }] }}}
```

Amazon provides a JSON data format to exchange defined interaction models, an excerpt is presented in Listing 1. It defines the invocation, its intents, as well as their slots and sample sentences. The complete JSON file can be downloaded as part of the project's source codes at GitHub: `https://github.com/fnogatz/alexa.pl`.

4.2 Prolog Web Server

As part of the skill creation it is required to state the address of the web service that handles the text-based user request. For security reasons, Amazon requires a valid SSL certificate for this endpoint.[6] Amazon uses HTTPS over port 443 to communicate with this web service. For every user request, an HTTP request with JSON body is sent to the Prolog web server. It contains information about the requested intent; the slot values are passed as a string. For instance, for the example sentence of Figure 3, the value of the `proveslot` as defined in the interaction model of Listing 1 is simply passed as the string `"Sam is a human"`.

As part of our contribution, we provide a basic Prolog web server based on SWI-Prolog's *library(http)*. It makes use of the SWI-Prolog version 7 extensions [10], as the recently added `dict` Prolog data type closely resembles JSON. Listing 2 shows the basic definition of the Prolog web server. It is a generic framework to parse HTTP requests and create JSON replies according to the data format as provided and expected by AVS.

[6] The libraries to support HTTPS in SWI-Prolog are still under development. We therefore use *nginx* (`https://www.nginx.com/`) as a reverse proxy. As a result, the described Prolog web server runs under port 8080.

All HTTP requests are dispatched to the Prolog predicate `alexa/1`, which first creates a dict from the request's JSON body. With the help of `get_intent/2`, this nested JSON document is transformed into a small Prolog term `IntentData` that holds the information about the intent and slot values.

As a result, for a new Alexa skill one only needs to define the appropriate `process_alexa_request/2`, that gives an answer as text depending on the passed `IntentData`. The JSON reply is then again built using dicts, and finally sent to Amazon's servers for user output.

Listing 2 Definition of the Prolog web server in alexa.pl. Only the implementation of `process_alexa_request/2` needs to be provided to support a new skill.

```
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/thread_httpd)).

:- http_server(http_dispatch, [ port(8080)]).
:- http_handler(/, alexa, [methods([get,head,options,post]),prefix]).

alexa(Request) :-
  http_read_json_dict(Request, DictIn),
  get_intent(DictIn, IntentData),
  process_alexa_request(IntentData, Answer),
  answer_dict(Answer, DictOut),
  reply_json(DictOut).
```

4.3 Connecting Alexa with RACE

For our example of using Alexa with RACE, we define `process_alexa_request/2` to forward the sentences to the RACE web service. This is done in the *model* in `alexa_mod.pl`. In particular, the current request (e.g., "prove that Sam is a human") has to be combined with the previously stated axioms. Once a new axiom is given by the user in a "remember" intent, its text-based paraphrase is stored in the service's *knowledge base*. For instance, given the previously stored axioms "Sam is a man" and "Every man is a human", it can be proven that Sam is a human.

4.4 library(race)

As the source code of RACE is not available, it can currently only be integrated into the Alexa skill via its web service (cf. Section 3.2). In order to access interfaces described in the *web service description language* (WSDL) standard, SWI-Prolog provides the add-on *library(wsdl)*.[7] On top of this, we built *library(race)*, a Prolog client for the SOAP interface of RACE. Its source code is available at GitHub: `https://github.com/fnogatz/race`.

library(race) defines the three Prolog predicates `check_consistency`, `prove`, and `ask`. In Listing 3, an example query is presented. RACE returns two proofs. In the first answer, "who" is substituted by "Sam". The second solution is deduced because there is a human if there is a at least one man, which could be proven by knowing that Sam is a man. In our service's model we select the best answer to give to the skill's user. The parts of the proof are combined into a single sentence, as shown in the example of Figure 3. The sentence has a generic form and integrates all `substitution/2` terms as returned by *library(race)*.

[7] SWI-Prolog package *wsdl* – `http://www.swi-prolog.org/pack/list?p=wsdl`

Listing 3 Usage example for `ask/4` of *library(race)*.

```
%% ask(+Knowledge, +Question, -Result, +Options)
?- ask("Every man is a human. Sam is a man.",
       "Who is a human?", Result, []).
Result = results([
  proof([ fact("Every man is a human."),
          fact("Sam is a man."),
          substitution("who", "Sam") ]),
  proof([ fact("Every man is a human."),
          fact("Sam is a man."),
          substitution("who", "(at least 1) man") ]) ]).
```

Using the SOAP interface of RACE requires an additional network round-trip time for every interaction with the skill. In addition to just send the speech to AVS, our Prolog web server has to forward the text-based message to the RACE web service. As a result, the response time – which is critical in speech-based natural user interfaces – gets longer. This problem becomes even more serious because the underlying SOAP protocol is stateless, i.e., the whole knowledge base has to be sent to the RACE web service as part of each request, resulting in even bigger messages and longer round-trip times.

5 Conclusions and Future Work

Voice-controlled systems become more and more available – both for consumers and developers. With its *Developer Platform*, Amazon opens the new field of speech-based natural user interfaces to a broad community of developers which are not familiar with NLP. But although the specification of new commands using example phrases and slots is easy to use, it is also very limited. We herein present an integration of Amazon's Alexa with the controlled natural language ACE, which allows us to support more flexible user queries. As a proof of concept, our skill can be used to formulate and store knowledge in natural language, propose questions, and prove theorems. It provides a Prolog web server that can be adopted for multiple different Alexa skills as well, and *library(race)*, an add-on for SWI-Prolog to use the SOAP web interface of the ACE Reasoner.[8]

Field testing showed that the speech-to-text translation of AVS becomes fuzzy for longer input sentences. With simple slots, their values are very restricted, not least because Amazon provides several pre-defined slot data types. This is not the case for long inputs such as `AMAZON.SearchQuery`. As the Attempto Parsing Engine also translates text, a better recognition rate might have been achieved for AVS returning the best n sentences instead of just one.

We now plan to further experiment on platforms other than Alexa, resorting for instance to Google Assistant (a.k.a. Google Home) or Apple Siri, as far as these provide APIs which allow for high-level extensions such as that which we explored in the present work. One development we are looking forward to work on is to build a prototype based on the speech recognition software Mycroft [5] running on a PC with Linux, thereby achieving full control of the entire software stack. Some components of the current integration can be easily re-used, e.g., the Prolog web server, and *library(race)*.

[8] The source codes are available at `https://github.com/fnogatz/alexa.pl` and `https://github.com/fnogatz/race` (MIT License).

References

1 Norbert E. Fuchs. First-Order Reasoning for Attempto Controlled English. In *International Workshop on Controlled Natural Language*, pages 73–94. Springer, 2010.

2 Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In *Reasoning Web*, pages 104–124. Springer, 2008.

3 Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In *FLAIRS Conference*, volume 12, pages 664–669, 2006.

4 Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English (ACE) Language Manual Version 3.0. `http://attempto.ifi.uzh.ch/site/pubs/papers/ace3manual.pdf`, 1999.

5 W. Wayt Gibbs. Build your own Amazon Echo – Turn a PI into a voice controlled gadget. *IEEE Spectrum*, 54(5):20–21, 2017.

6 Stefan Hoefler. The syntax of Attempto Controlled English: An abstract grammar for ACE 4.0. Technical Report ifi-2004.03, Department of Informatics, University of Zurich, 2004.

7 Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer, 2013.

8 Irene Lopatovska, Katrina Rink, Ian Knight, Kieran Raines, Kevin Cosenza, Harriet Williams, Perachya Sorsche, David Hirsch, Qi Li, and Adrianna Martinez. Talk to me: Exploring user interactions with the Amazon Alexa. *Journal of Librarianship and Information Science*, page 0961000618759414, 2018.

9 Gustavo López, Luis Quesada, and Luis A Guerrero. Alexa vs. Siri vs. Cortana vs. Google Assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250. Springer, 2017.

10 Jan Wielemaker. SWI-Prolog version 7 extensions. In *Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments*, pages 109–123, 2014.