Electronic version of an article published in
International Journal of Semantic Computing
Vol. 12, No. 3 (2018) 335-360
https://doi.org/10.1142/S1793351X18400159
© World Scientific Publishing Company
https://www.worldscientific.com/worldscinet/ijsc

Detection of Control Structures in Spoken Utterances

Sebastian Weigelt, Tobias Hey, Vanessa Steurer

Karlsruhe Institute of Technology
Institute for Program Structures and Data Organization
Karlsruhe, Germany
weigelt@kit.edu, hey@kit.edu, vanessa.steurer@student.kit.edu
http://ps.ipd.kit.edu

State-of-the-art intelligent assistant systems such as Siri and Cortana do not consider control structures in the user input. They reliably react to ordinary commands. However, their architectures are not designed to cope with queries that require complex control flow structuring. We propose a system to overcome these limitations. Our approach models if-then-else, loop, and concurrency constructs in spoken utterances explicitly. The model bridges the gap between linguistic and programmatic semantics.

To demonstrate our concept, we apply a rule-based approach. We have implemented three prototypes that use keyphrases to discover potential control structures depending on the type of control structure. However, the full structures are determined differently. For conditionals we use chunk and part-of-speech tags provided by natural language processing tools; for loops and concurrency we make use of an action extraction approach based on semantic role labeling. Additionally, we use coreference information to determine the extent of the respective structure.

The explicit modeling of conditionals, loops, and concurrent sections allows us to evaluate the accuracy of our approaches independently from each other and from other language understanding tasks. We have conducted two user studies in the domain of humanoid robotics. The first focused on conditionals. Our prototype achieves F_1 scores from 0.783 (automatic speech recognition) to 0.898 (manual transcripts) on unrestricted utterances. In the second the prototypes for loop and concurrency detection also proved useful. F_1 scores range from 0.588 (automatic speech recognition) to 0.814 (manual transcripts) for loops and from 0.622 (automatic speech recognition) to 0.842 (manual transcripts) for concurrent sections respectively.

Keywords: Spoken Language Understanding; Naturalistic Programming; Spoken Language Interfaces; End-User Programming.

1. Introduction

Intelligent assistants such as Siri, Google Assistant, Alexa, and Cortana have been the latest trend in user interfaces [1, 2]. Although these systems appear rather smart today, there is still much room for improvement. Most notably, intelligent assistants struggle with conditional expressions, such as "Book a business class flight to LA. If no business class tickets are available book economy." As these systems are built to process single requests, they will most likely fail to interpret such utterances correctly. In the above example an intelligent assistant must understand, that the user wants the system to execute only one of the two potential actions. Furthermore, the choice depends on a condition that is expressed in the utterance. Presumably, this type of query is only the beginning; users will soon expect intelligent assistants to cope with more complex control flow structures, i.e., loops and concurrency. To fully understand control structures a system has to discover statements that imply a change in control flow together with their dependent statements, i.e., the actions that depend on a condition or are supposed to be executed multiple times. The placement and length of the dependent statements is hard to predict. Often even humans cannot decide whether a statement is dependent or independent. E.g., if we add, "also order a Wi-Fi pass for me," to the above example, it is unclear whether the ordering of the Wi-Fi pass depends on the availability of business class tickets.

To deal with control structures a system needs a dedicated approach to process statements that manipulate the control flow in spoken language. The advantages of such a treatment are:

(a) the development process is single-minded and thus presumably less error-prone
(b) the evaluation of the approach can easily be conducted intrinsically and extrinsically
(c) the solution can be replaced by another without influencing other analyses

We propose to infer the semantics of conditionals, loops, and concurrency from spoken utterances. Our approach focuses on dependent clauses. In contrast to previous work on control structures, it is capable to determine dependent clauses that span multiple statements reliably. Our approach is part of the research project *PARSE*, which aims at end-user programming in natural language (PNL). The associated framework is extensible, so-called *SLU agents* can be added to improve *PARSE*'s language understanding abilities. To separate concerns we have implemented independent agents for each type of control structure. With the aid of the presented approach *PARSE* is capable of synthesizing scripts that contain control structures like the one depicted in Figure 1. The conditional statement starting with *if the laundry* is mapped to an if-then-else construct. In the same manner, the signal word *twice* is interpreted as a for-loop inside the then-block. Finally, our approach synthesizes a repeat-until (from *wait until the laundry is done*) and a concurrent section (from *while you take the laundry from the washer check its condition*).

The remainder is structured as follows: First, we discuss related work in Section 2 and introduce the project *PARSE* in Section 3. Then we discuss the linguistic fundamentals we need for our approach in Section 4. Subsequently, we present the implementation of three prototypes that generate semantic representations of conditionals, loops and concurrent sections from spoken utterances (Section 5).

Input: *after dinner go to the laundry room and start the washer wait until the laundry is done while you take the laundry from the washer check its condition if the laundry is dry iron the shirts and fold them twice otherwise put the laundry into the dryer*

Script:

```
robot.goTo($laundryRoom$);
robot.start($washer$);
repeat
    robot.wait();
until $laundry.done = true$
do together
    robot.empty($washer$);
    robot.checkState($laundry$);
end do together
if $laundry.dry = true$ then
    robot.iron($shirts$);
    for $int\, i = 0$; $i < 2$; $i{+}{+}$ do
        robot.fold($shirts$);
    end for
else
    robot.put($laundry$, $dryer$);
end if
```

Fig. 1: Input/output example: a script synthesized by our tool.

Finally, we discuss evaluation results for manual transcripts and the output of two automatic speech recognition systems (ASR) in Section 6 before we conclude the paper (Section 7).

2. Related Work

The processing of control structures is inevitably for PNL. However, the way control structures are processed differs remarkably in terms of how explicit they are modeled and treated:

Approaches that aim at code dictation, such as AppleScript [3], Natural Java [4], or Spoken Java [5, 6] do not consider the ambiguity of control structures. Since the user has to enter code-like descriptions these approaches leave it up to the user to properly structure if-then-else or loop constructs. Thus, these approaches evade challenges like reversed conditional structures or the determination of the dependent sections. In the main, they use keyword matching and heuristics on sentence structures to identify control structures.

In the area of end-user PNL systems expect a behavior description rather than code dictation. Thus, they have to translate conditionals and loops expressed in natural language into programmatic counterparts. Most approaches integrate control structures into their language model inherently. E.g., the tool SmartSynth by Le et al. [7] is an approach for synthesizing scripts for smartphones from written texts. It detects domain specific keywords and type-based constructs to infer script elements and data flow. Conditionals play a major role. SmartSynth's language model expects a condition at the beginning of a script followed by an action that is triggered if the condition is true. The more recent work on semantic parsing by Quirk and Beltagy [8, 9] has a similar objective. Their approach synthesizes so called if-this-then-that recipes from single statements. As a consequence, both approaches do not have to decide whether a condition is part of the script or not. Additionally, the dependent clause is simply everything that is stated after the condition. Vadas and Curran [10] propose to use Combinatory Categorial Grammars (CCG) [11] to synthesize python code from unrestricted natural language. Their rule-based approach converts sub-trees of CCG derivations into a hand-crafted logical representation. This representation can also represent basic if-then-else structures and loops. The representation of conditionals is also part of other language formalisms such as LFG [12] or HPSG [13]. However, they focus on the lexical structure. Thus, determining the dependent clauses on (potentially ungrammatical) spoken input is out of scope. Landhäußer and Hug [14] analyzed the usages of loops and concurrency in written natural language. In a preliminary study they figured out that these control structures are described radically different in natural language: E.g. one might say, "Do A three times," or, "Do A. At the same time do B." The first statement is an implicit description of a loop, the latter of parallel execution. In a prototypical implementation they used keywords and heuristics on dependency graphs to synthesize control structures. The tool Metafor [15] by Liu and Lieberman is intended to synthesize code stubs in Python from narrative descriptions. Later it was extended by Mihalcea et al. [16]. Instead of stubs Metafor now synthesizes executable Python code. Additionally, they added support for if-then-else constructs and loops. To synthesize code elements they use a customized parser that generates Subject-Verb-Object-Object structures. Later, the SVOO structures are mapped to objects and methods. With the help of keywords and language patterns they infer whether (conditionals) and how often (loops) these structures shall be executed.

3. PARSE

Our work on detection of control structures in spoken utterances is part of the project *PARSE* [17]. The goal of the project is to enable a layperson to program in plain spoken English. Typical application areas of *PARSE* are robotics, home automation, and the like. To facilitate programming with spoken language the system must understand the user's intents. Thus, *PARSE* is actually a system for spoken language understanding (SLU) [18]. To achieve deep SLU *PARSE* takes the

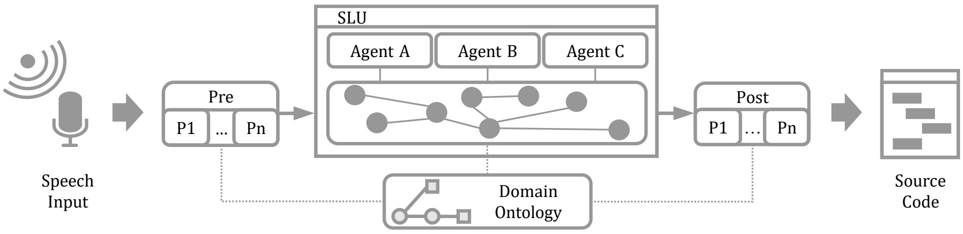

Fig. 2: The architecture of *PARSE*.

approach of independent agents. Every agent is responsible for a specific SLU task. As SLU tasks are interdependent in general, all agents work in parallel and therefore may benefit from results of other agents. The strict separation of concerns enables us to either build an agent knowledge-based or probabilistically according to the SLU task at hand and evaluate it intrinsically. The architecture of *PARSE*, which is illustrated in Figure 2, is separated in three independent parts: a pipeline for pre-processing, an agent-based main execution, and a pipeline for post-processing. A graph serves as shared data structure for the agents. The pre-processing pipeline is meant for common natural language processing tasks, e.g., automatic speech recognition, shallow parsing, and named entity recognition. The user's utterance is processed sequentially here. In the last pre-processing step, the initial graph is built and passed to the main execution module. The main execution is responsible for SLU. There, agents for deep SLU work in parallel and transform the graph to publish their results. That way, a semantic representation of the input is created incrementally. SLU tasks encompass extraction of actions and topics, context [19] and coreference analysis, or—as proposed here—the detection of conditionals, loops and concurrent sections. If the agents are unable to transform the graph into a proper intention model, the utterance is likely to be incomplete or ambiguous. In such situations the user is queried for clarification [20]. The post-processing pipeline performs the steps required to map the user's intent—modeled in the graph—to functions of the target system. Target systems are modeled in ontologies as proposed in our previous project *NLCI* [21]. We have also shown, that domain ontologies can be extracted (semi-)automatically from most APIs with small effort.

4. Linguistic Foundation: Conditionals in Natural Language

Conditional branching is the base for any kind of control flow manipulation. In common programming languages it is the integral part of if-then-else structures and loops. These constructs are partially derived from their counterparts in natural language. Humans often use conditionals to structure the discourse or to establish dependencies between statements. Repetition is also used in natural language. For example, when humans instruct each other, they use phrases such as "until you are done" to express that some events or actions are supposed to be conducted repeatedly *until* a condition is fulfilled.

In natural language conditionals are stated less formal. However, the syntactic structure of conditionals follows grammatical rules. The syntactic form of a conditional also influences its semantics and its impact on the discourse. Despite their importance in natural language there is no commonly shared definition for *conditionals*. Since many different forms of conditionals can be framed, only generic definitions are feasible. We refer to the one given by Declerck and Reed [22]:

> "A conditional is a two-clause structure in which one of the clauses is introduced by *if* (possibly preceded by *only*, *even* or *except*) or by a word or phrase that has a meaning similar to *if*, *only if* (e.g., *provided*) or *except if* (viz. *unless*)."

Haegeman additionally distinguishes two types of conditionals: event-conditionals and premise-conditionals [23]. Conditional clauses of event-conditionals express events (or states) that will lead to an event which is stated in the main clause. Premise-conditionals structure the discourse. In the following examples (1) is an event- and (2) is a premise-conditional:

(1) If it rains we will all get terribly wet and miserable.

(2) If [as you say] it is going to rain this afternoon, why don't we just stay at home and watch a video?

We expect users of PNL systems to form descriptive utterances or commands. Most of the conditionals used in descriptions or instructions are event-conditionals. Thus, our approach will focus on event-conditionals.

In the following subsections we discuss the syntax, semantics, and pragmatics of conditionals.

4.1. *Syntax*

Conditionals are two-clause structures; the clause that contains the condition is called *conditional clause*, whereas the dependent clause is called *main clause*. Most commonly, the main clause is composed of a then-clause, depicting an event that will occur only if the condition is fulfilled, and an optional else-clause. The latter describes what happens if the condition is not fulfilled. Syntactically, there are two ways to structure the clauses of conditionals:

(a) the conditional clause, followed by a then-clause and an optional else-clause: if-then(-else)

(b) a then-clause, followed by the conditional clause and an optional else-clause: then-if(-else)

However, the if-then(-else) structure is more commonly used.

4.2. *Semantics*

The semantics of conditionals are often derived from their so-called tense pattern [22]. In linguistics four tense patterns for conditionals are commonly used. For example, a conditional in tense pattern *1* has a conditional clause in simple past and a main clause in will future. This pattern is used to express hypothetical statements such as, "If you stay outside, you will get cold." We found that none of the patterns fits conditional descriptions or instructions. Here the conditional clause may be in simple present, will future, simple past, or even present perfect. However, we expect the main clause to be in simple present, which is the common shape of instructions (in imperative mood) and statements.

4.3. *Pragmatics*

Conditionals are used in different contexts and bear manifold meanings. Thus, there is no universally valid model for intent understanding. If we focus on the domain of PNL we can reduce the possible meanings. Assuming that all conditionals can be translated to if-then-else structures the intent model of conditionals can be described as follows.

First, a conditional clause must be present. Conditional clauses can be expressed in various ways, i.e., the same intention can be uttered in different syntactical structures. As stated by Declerck and Reed [22] a condition is introduced by a keyword. However, a variety of synonyms can introduce a condition. In the same way, specific phrases may imply a condition, like in, "the longer you rinse it, the more it will get clean." On the contrary, some clauses formed with if or its synonyms do not shape a conditional at all, as in, "I do not know if he really does it on purpose." Besides the conditional clause, the intended reference frame of the conditional must be determined. The reference frame characterizes the boundaries of the conditional structure, in particular:

(a) the extent of the conditional clause
(b) the position (before or after the conditional clause) and extent of the then-clause
(c) the existence and extent of the else-clause

The conditional clause encompasses a keyword and one or more phrases. Usually it ends with the beginning of the then-clause (if the then-clause follows the conditional clause). However, then-clauses can be introduced by a keyword as well as without. Also it might be placed either in front of the conditional clause or after it. Thus, the separation of conditional and then-clause is not obvious in all cases. The maximal extent of the then-clause might be limited by an else-clause. But as discussed before, the else-clause is optional. However, if an else-clause exists it is always introduced by a keyword or the negation of the condition, e.g., *if not*. From a syntactical point of view, its extent is unrestricted. Thus, its end can only be determined pragmatically. These uncertainties make it impossible to derive the pragmatic of conditionals from

$\boxed{\textit{if}}$ *I leave do the laundry* $\boxed{\textit{else}}$ *assist me with cooking*

Fig. 3: Expected user input to the system.

the syntactical structure solely.

5. Detection of Control Structures

In the following we describe the rationale behind our approaches to detect conditionals, loops, and concurrency. As described in Section 3 and Section 4 we expect control structures uttered in spoken language. Since *PARSE* is a framework for PNL user utterances are descriptions or instructions. Those utterances may contain one or more control structures or none at all. The wording and length of the utterances is unrestricted.

Transcriptions of spoken utterances, either generated by hand or with an ASR, usually lack punctuation. However, punctuation bears information about the intended control flow, as shown in the following examples:

(3) Clean the kitchen. [If I am still at work $_{\text{IF}}$] [do the laundry $_{\text{THEN}}$].

(4) clean the kitchen if I am still at work do the laundry

In (3) the dependent clauses of the conditional can be determined clearly due to the present punctuation. Without punctuation, such as in (4), the reference frame of the conditional becomes unclear. Now, “clean the kitchen,” may be the then-clause of the expression or an independent preceding sentence. Likewise, the phrase, “do the laundry,” can be interpreted differently. It either forms the then-clause or an independent instruction.

Besides the missing punctuation, spoken language commonly features hesitations, such as *uhm* or *ehm*. Additionally, ASR transcriptions introduce word errors. Thus, the approaches must be as robust as possible.

5.1. *Conditionals*

Our approach to detection of conditionals is grounded on the grammatical features discussed in Section 4. Figure 3 shows an expected user input. Note that conditionals of this sort might be located anywhere in the utterance.

If we have a closer look at this example, it becomes clear that keywords (in boxes) play an important role for the identification of conditionals. As discussed in Section 4 the conditional clause (green) is introduced by the word *if* or a word or phrase with a similar meaning. Similarly, the else-clause (orange) begins with the word *else* or a synonymous phrase or word. The then-clause (blue) in the example is not introduced by a keyword (but can be introduced by *then* or a synonym). If we additionally take into account that a conditional can appear anywhere in the

[If$_{\mathrm{SBAR}}$] [I$_{\mathrm{NP}}$] [leave$_{\mathrm{VP}}$] [do$_{\mathrm{VP}}$] [the laundry$_{\mathrm{NP}}$]

Fig. 4: Example sentence with chunk tags.

input and else-clauses are optional, it becomes apparent that an approach based on keywords or string matches is insufficient. Instead, the sentence structure might be useful to infer the reference of the condition. E.g., usually the condition keyphrase is followed by a noun and an imperative verb. If another imperative verb follows (and is not linked to the previous, e.g., with a conjunction) we can infer that the second verb already belongs to the then-clause. This is the case for "do the laundry" in Figure 3.

We have implemented our prototype as an agent for *PARSE*. Thus, we can utilize all results provided by its pre-processing pipeline and other agents. The pre-processing pipeline provides—among other results—part-of-speech (POS) and chunk tags. In computational linguistics chunks refer to a flat representation of the sentence structure composed of its constituents. Figure 4, shows a sentence with chunk tags: the tag SBAR refers to a subordinating conjunction, NP to a noun phrase, and VP to a verb phrase.

Additionally, another agent offers reference analysis information. It adds coreference and identity information to *PARSE*'s graph representation. E.g., given the utterance, "if the laundry is dry iron it and fold the laundry afterwards," the agent infers that both occurrences of *laundry* and the *it* all refer to the same entity `laundry`.

Our prototypical implementation uses a set of grammars to infer the structure of conditionals. All production rules are derived from the linguistic characteristics of conditionals discussed in Section 4.

Our approach processes the input in two phases. First, we locate basic conditional structures with a keyword search and a simple grammar that uses chunk and POS tags. Second, we determine the reference frames of the conditionals. In this step we expand the basic structures utilizing advanced grammars and reference information.

5.1.1. *Basic Conditional Structure*

To identify basic conditional structures we first search for keywords. The keywords we used for our approach are listed in Table 1. We use the grammar depicted in Figure 5 to build the basic structures around the keywords. If the grammar can not produce a valid tree for the keyword, the keyword is discarded. The result of the first step is a candidate set of basic if-, then-, and else-structures. Note that to this point the structures are not connected. Since many then-clauses are not introduced by a keyword we miss them at this point. The basic structures represent the simplest way to express conditionals. Hence, they can be seen as the minimal construct our approach is able to extract from the input.

Table 1: The list of used keyphrases for the detection of conditionals.

Clause Type	Keywords/-phrases
conditional clause	if, when, suppose(d) that, supposing that, whenever, in case, in the case that, unless, on condition that, providing that, provide(d) that, as long as, else if
then-clause	then, please, if so, you can, you have to, could you, would you
else-clause	else, if not, otherwise, elseways, alternatively, instead, either, rather, oppositely

$$\begin{array}{rcl}
\text{conditional} & \rightarrow & \text{if-clause then-clause} \\
\text{conditional} & \rightarrow & \text{if-clause then-clause else-clause} \\
\text{if-clause} & \rightarrow & \textit{if-keyword NP VP} \\
\text{then-clause} & \rightarrow & \textit{then-keyword VP} \mid \textit{VP} \\
\text{then-clause} & \rightarrow & \textit{then-keyword NP VP} \mid \textit{NP VP} \\
\text{else-clause} & \rightarrow & \textit{else-keyword VP} \\
\text{else-clause} & \rightarrow & \textit{else-keyword NP VP}
\end{array}$$

Fig. 5: Context-free grammar describing the basic structures.

The rationale behind the production rules for the basic structures is as follows: The basic structure of the if-clause consists of the keyword itself followed by a noun phrase and a verb phrase. The entity that the condition refers to resides in the noun phrase. In a minimal setting the noun phrase is succeeded by a verb phrase only. Such a minimal example would be, "if I leave." The basic structure of a then-clause can be formed in two ways, depending on the voice (or mood) of the utterance. First, for the imperative mood, a verb phrase is sufficient to express a command, like in, "if I leave *stop*." Second, for descriptive utterances in active voice a sequence of a noun phrase followed by a verb phrase is needed, e.g., "if I leave *the robot must stop*." The same applies to the else-clause. Therefore, we can reuse the production rules for the then-clause. However, contrary to the then-clause, a keyword is mandatory. This is the case because the else-clause is optional. Therefore, it must be introduced by a keyword to adhere to the English grammar.

As mentioned before, we miss all then-structures, which are not introduced by a keyword. Because omitting the keyword is more common, we added a special treatment for then-clauses that does not rely on the keyword search. To identify missing then-clauses we process all consecutive if- and else-structures. We apply the production rules for then-clauses to the phrases between these structures. If a valid structure can be extracted, it is added to the set of candidates for basic then-structures. Finally, we assemble the candidates using the first two production rules of the basic grammar. Thus, we end up with a candidate set of basic if-then(-else)

if-clause → *if-keyword* NBS
NBS → NPB VPB | NPB VPB *Conj* NPB VPB
NPB → *NP* CC NPB | *NP PP* NPB | *NP*
VPB → *VP* CC VPB | *VP PP* NPB| *VP PP* VPB | *ADVP* VPB |
VP VMD | *VP* VMD CC VPB | *VP*
CC → *Conj* | *Neg*
VMD → *ADJP* | *ADJP* CC VMD | *ADVP* | *ADVP* CC VMD | *PRT* |
PRT CC VMD

Fig. 6: Excerpt of the context-free grammar used for if-clauses.

structures.

5.1.2. *Reference Frame*

After the identification of basic structures, we infer the reference frames of the conditionals. To do so, we expand the candidates using two different techniques:

(a) syntactically, using additional grammars and
(b) pragmatically, using (co-)reference and identity information.

Both techniques were designed conservatively, i.e., we try to extend the basic structures but ensure that we do not include statements which were intended to be independent. We decided to focus on precision rather than recall to mimic human behavior. As mentioned in Section 1 even humans struggle with determining the reference frame. However, in unclear situations humans tend to assume that a statement is independent rather than dependent.

Syntactical Extension To extend the basic structures we use additional grammars in a first step. These grammars can produce expanded if-, then-, and else-structures. The most important production rules for the if-grammar are illustrated in Figure 6. We do not apply a common NLP parser to identify the clause structures for several reasons. First, NLP parsers are designed to process written text. Thus, they struggle with ungrammatical language and missing punctuation. Second, NLP parsers usually provide a full parse. Instead we are interested in partial parses, that are robust and precise. Third, by defining our own grammar, we are able to integrate structural characteristics of conditional clauses into the production rules. The production rules are derived from grammatical characteristics. First of all, some syntactic structures link phrases. Conjunctions and negations create homogeneous blocks of noun phrases and verb phrases. We call this type of consolidation *noun phrase block* (NPB) and *verb phrase block* (VPB). Examples are “the laundry *and* the lunch” or “eat *not* drink”. Note that the production rules for verb phrase blocks encompass additional linking structures like adverbial phrases and prepositional phrases. The latter can also link verb phrases and noun phrase

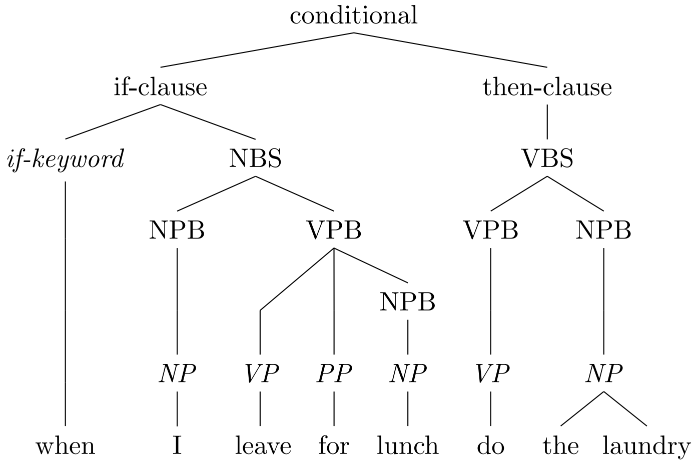

Fig. 7: Parse tree for “when I leave for lunch do the laundry”.

blocks to produce inhomogeneous verb phrase blocks. We can join the phrase blocks to form so called *noun block structures* (NBS) and *verb block structures* (VBS) depending on the order of blocks. The context-free grammar for if-clauses only has a NBS production rule because one can not begin an if-clause with a verb phrase (or a verb phrase block). The grammars for then- and else-clauses are composed of more production rules. Also the definition of phrase blocks differs slightly. If we cannot expand a candidate structure, we fall back on the basic structure. Figure 7 shows the parse produced by the grammar for if-clauses (Figure 6) and then-clauses (not depicted here).

Extension using references As discussed before, our approach to extend the basic structures based on syntactical features is rather conservative. Thus, we found that we need an approach that captures the user's intent more accurately. As a first step towards intention extraction for conditionals, we focus on coreference and identity information. The basic idea is as follows. If the same entity is mentioned in consecutive phrases, it is very likely that the phrases belong to the same type of clause. The example in Figure 8 illustrates the idea.

The technique presented in the previous paragraph identifies “do the laundry” as the then-clause. However, the following phrases “iron it and fold it” are most likely intended to belong to the then-clause as well.

To handle such cases we extract all coreference relations provided by *PARSE*'s agent for coreference analysis. We consider all so-called coreference chains (in the example: it → it → laundry). As spoken utterances yield no reliable information about sentence boundaries we are not able to distinguish between inter- and intra-

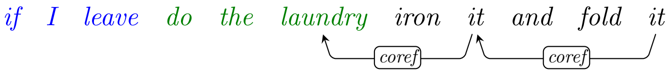

Fig. 8: Example illustrating useful coreference information.

sentential coreference. Thus, we treat all references alike. We add all phrase blocks, which are part of the chain, to the clause that contains the first element. In the example we would expand the then-clause by the phrase blocks "iron it" and "fold it". Of course, we split up coreference chains that span basic structures of different type. Using coreference chains our approach can reliably determine reference frames of multiple statements.

5.2. *Concurrency & Loops*

For our approach on concurrency and loops we can draw less from linguistic theory. Repetition and concurrency are not regularly used in natural language. Also, the semantics are manifold and often ambiguous. Consequently, there are no commonly accepted syntactical constructs to express these kinds of semantic structures. However, if we want to provide PNL, we have to deal with concurrency and repetition; sooner or later users will expect that they can use these constructs to create longer and more complex commands. As loops usually contain some kind of termination condition, we can reuse the approach for detecting the conditional clause (see Subsubsection 5.1.1).

We used the work of Landhäußer and Hug [14] as orientation. Their approach follows the intuition that control structures are introduced by keyphrases. Sometimes the keyphrase reveals the type of the control structure; in other cases an in-depth analysis of the context is needed. This includes the number and order (and placement) of the dependent clauses. Landhäußer and Hug also note that a keyphrase may bear additional information. E.g., if one says, "do A twice!" *A* is supposed to happen two times, i.e., the keyword *twice* not only indicates a for-loop but also an iteration count of two.

However, their approach heavily depends on written input; the syntactical patterns they use to detect control structures are based on dependency graphs (provided by Stanford CoreNLP[a], which does not work properly for spoken input) and punctuation, which is not available as discussed in Section 5. Also, we came to the conclusion that some keyphrases are not used properly and some assumptions they made (regarding the sentence structure) are not valid in general.

Thus, we developed an all-new approach. We also use keyphrases. However, we assign types to keyphrases. The type determines how we perform the following syntactical analysis of the dependent phrases. Instead of chunk tag patterns we use

[a] https://stanfordnlp.github.io/CoreNLP/

Table 2: The list of keyphrases used for concurrency detection.

Type	Keywords/-phrases
Wrapping	at once, simultaneously, coevally, concurrently, synchronistically, during
Separating	at the same time, while, whilst, meanwhile, in the meantime, while doing so
Opening	and while, while, and whilst, whilst
Ending	at the same time, in the meantime, while doing so

information about the performed actions provided by another *PARSE* agent. This agent infers the relationships between actors, actions, and objects (i.e., parameters) in the utterance based on semantic role labeling (SRL).

To detect concurrent sections and loops we partially parse the words around keyphrases for fitting actions. We use adapted rule sets for different types of keyphrases. Finally, we examine the number of affected actions to determine the extent of the dependent clauses. We will detail the approaches for concurrency and loops—both implemented as *PARSE* agents—in the upcoming subsections.

5.2.1. *Concurrency*

As discussed in Subsection 5.2 there is no linguistic grounding for concurrency. Therefore, we define concurrency as a structure consisting of two or more clauses (referred to as *concurrent clauses*). Each clause encompasses one or more events (i.e., actions in the context of end-user programming) that (are supposed to) happen simultaneously. The programmatic counterparts are code sections that can be processed by multiple threads in parallel, e.g., the OpenMP pragma *omp parallel sections*. In natural language we expect that concurrent clauses are indicated by certain keyphrases. E.g., one might say, "do A *while* you do B" or "*simultaneously* do A and B." The examples show that the clause structure depends on the keyphrase. In the first example the keyphrase separates the two concurrent clauses while it opens the clause structure in the second.

To detect concurrent sections we proceed as follows. First, we extract keyphrases. We define four different types of keyphrases:

- **Wrapping**: The concurrent clauses can either be placed before or after the keyphrase. However, we assume that after the keyphrase is more common.
- **Separating**: The keyphrase is placed between the concurrent clauses.
- **Opening**: The keyphrase precedes the concurrent clauses.
- **Ending**: The keyphrase succeeds the concurrent clauses.

The keyphrases and their respective types are listed in Table 2. Note that some keyphrases have two types. E.g., *while* has the primary type *Separating* and the

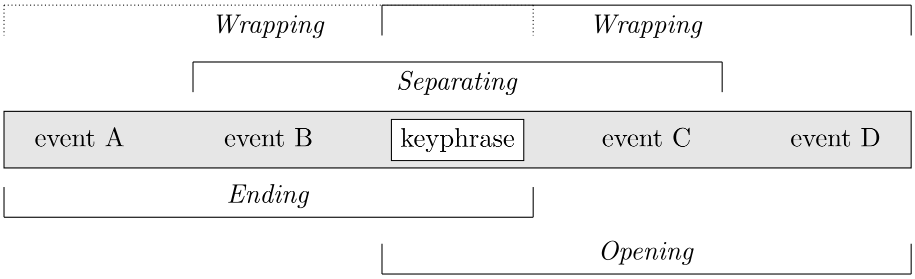

Fig. 9: Overview of the spans of the concurrency keyphrase types.

secondary type *Opening*; the order of the types corresponds to the row number in the table.

After we have extracted the keyphrases, we proceed with the detection of the dependent clauses. As discussed before, we cannot use a parser or dependency parser to determine dependent clauses. Instead, we use a *PARSE* agent that detects actions (along with the actor and parameters, specified by means of wh-words, i.e., who, what, where, when, and how). The detection is based on the SRL tool SENNA [24]. SRL is challenging as such (SENNA achieves F_1 75.49% for CoNLL-2005 Shared Tasks on Semantic Role Labeling [25]), on spoken input the performance decreases further. In a preliminary study SENNA proved to be more robust than others in our domain. Also, SENNA is relatively fast, which is a critical requirement in an interactive setting. To improve the quality of the SENNA's results, we added some heuristics to recover from typical misclassifications on spoken language.

We determine concurrent clauses according to the keyphrase type. Figure 9 illustrates the relation between keyword type and extracted clauses. We decided to determine clauses conservatively, i.e., we extract only one event per clause, because without punctuation anything else is close to guessing. E.g., if one states, "do A do B *while* you do C and D," it is unclear whether only the events B and C are supposed to happen at the same time, or A and B simultaneously to C and D. On the other side, we are able to combine and even nest multiple clauses as long as any pair of clauses has a keyphrase. E.g., from statements such as, "do A *while* you do B *in the meantime* do C," we can infer that A, B, and C are simultaneous events. Also, we are able to resolve some elliptic references with the help of the agent for action detection. Whenever two actors perform the same action (or whenever more than one parameter of the same type is involved in an action) the action is mentioned only once, e.g., "John and Mary turn" or "Get me an orange and an apple". In such cases the agent duplicates the predicate. Hence, we can determine the correct concurrent clauses for expressions like "John and Mary turn simultaneously".

If we cannot determine the concurrent clauses with the primary type of the keyphrase, we repeat the process with the secondary type. Whenever we are unable to extract two clauses per keyphrase, we discard it. We decided not to integrate conditional concurrent sections into our approach. We found out that they behave just like the nested version: an if-then-else structure with a concurrent section inside the then- or else-block. Therefore, we can simply let the agent discussed here and the one presented in Subsection 5.1 work independently.

5.2.2. *Loops*

Again, lacking a linguistic grounding, we define looping as follows. A loop is a structure consisting of two clauses, where one clause contains a condition and the other clause encompasses one or more events (i.e., actions, in the context of end-user programming) that (are supposed to) happen multiple times as long as a condition holds. The former clause corresponds to the termination condition of loops in programming languages. Subsequently, we call this type of clause *loop conditional clause*. The latter is the equivalent to (programming) statements inside the loop body, called *loop body clause* in the following. We expect that repeated events are indicated by keyphrases such as *while* or *three times*. The keyphrase usually resides inside the loop conditional clause and possibly bears additional information as discussed later.

In programming languages different types of loops are used. For our approach we focus on the detection of linguistic structures that can be mapped to while, do-while, and for (each) loops. These are commonly used in popular programming languages. Also, natural language counterparts exist for these loop types. However, repetition can be stated in many ways, which makes the mapping to program structures difficult. The exemplary statements, "turn *twice*," and "*while* the fridge is open take out beverages," where the first can be interpreted as a for loop and the second as a while loop, illustrate this issue. The term *twice* in the first example not only indicates a for loop but also the number of iteratons. The same applies to phrases like, "four times." In the second example the keyword *while* is followed by a condition statement (that determines the termination condition of the repetition). This extends to keyphrases such as *until*, *as long as*, and the like. Furthermore, the keyword *while* is ambiguous, since it is also a keyphrase for concurrency. For disambiguation, we examine the context of the keyphrase. If it is succeeded by condition statements it is most likely used to indicate a loop; otherwise it indicates concurrency.

To detect loops in spoken utterances we first search for keyphrases. If the condition is not inherently stated in the keyphrase, e.g., *twice* or *three times*, we perform a partial parse around the keyphrase, as we do for conditional clauses in Subsubsection 5.1.1. However, only a subset of the grammar rules can be applied here. For conditionals we allowed statements and events/actions as conditional clauses. Because of the ambiguity of keyphrases discussed above, only statements are legit termination conditions for loops. Thus, we use the grammar shown in Figure 10 to extract loop conditional clauses.

After we have extracted the keyphrases and conditions we categorize them. We use the same types like in Subsubsection 5.2.1 except *Separating*. Instead, we

$$\begin{array}{rcl}
\text{loop-conditional-clause} & \rightarrow & \textit{loop-keyphrase}\ \text{NBS} \\
\text{NBS} & \rightarrow & \text{NPB VPB} \mid \text{NPB VPB}\ \textit{Conj}\ \text{NPB VPB} \\
\text{NPB} & \rightarrow & \textit{NP}\ \text{CC NPB} \mid \textit{NP}\ \textit{PP}\ \text{NPB} \mid \textit{NP} \\
\text{VPB} & \rightarrow & \textit{VP}\ \text{VMD} \mid \textit{VP}\ \text{VMD CC VPB} \\
\text{CC} & \rightarrow & \textit{Conj} \mid \textit{Neg} \\
\text{VMD} & \rightarrow & \textit{ADJP} \mid \textit{ADJP}\ \text{CC VMD} \mid \textit{PRT} \mid \\
 & & \textit{PRT}\ \text{CC VMD}
\end{array}$$

Fig. 10: The context-free grammar used for *loop conditional clauses.*

Table 3: The list of used keyphrases for loop detection ([CD] is a wildcard for any word with POS tag CD (cardinal number)).

Type	Keywords/-phrases
For	twice, thrice, [CD] times
Wrapping	while, as long as, until, till
Opening	and while, and as long as
Ending	repeat this until, repeat it until, do it until, do this until

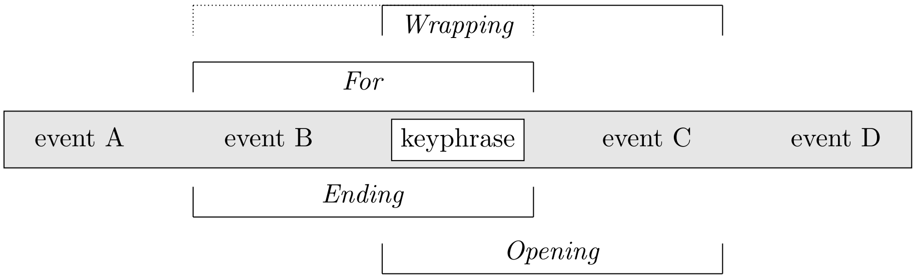

Fig. 11: Overview of the spans of the loop keyphrase types.

define the type *For*, where the keyphrase hints at a for loop structure and indicates the number of iterations. The loop body clause precedes the keyphrase in that case. Table 3 lists the keyphrases we use for loop detection and their respective types. Note that unlike the concurrency keyphrases all keyphrases for loops have an unambiguous type.

The detection of the dependent clause (the loop body clause) is similar to the respective process for concurrency. Again, the choice of considered events depends on the type of keyphrase and is illustrated in Figure 11. The *loop body clause* can be placed before or after the keyphrase. Note that without punctuation we can only determine *loop body clauses* that encompass one event (for similar reasons as discussed in Subsubsection 5.2.1).

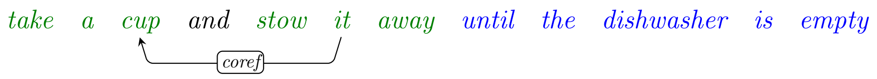

Fig. 12: Example illustrating useful coreference information for loops.

5.2.3. *Extension using references*

The approaches presented in Subsubsection 5.2.1 and Subsubsection 5.2.2 are rather conservative; they restrict the number of events per concurrent clause to two and per loop body clause to one. To detect longer clauses we adopt the approach of extension of clauses by means of coreference information (see Subsubsection 5.1.2). The approach is grounded on the intuition that events, which affect the same objects or places more likely belong together than others. Thus, we can determine identical objects (or places) by tracing coreference chains as described in Subsubsection 5.1.2. If coreference spans a keyphrase, we break up the chain. An example is shown in Figure 12. There, the basic approach for loops determines only the event “stow it away” as loop body clause. With the help of coreference information we can add “take a cup” to the clause.

6. Evaluation

To evaluate our approach to detection of control structures in spoken utterances we conducted two user studies. The first involved nineteen participants and focused on conditionals, the second on loop and concurrency detection and was conducted with ten participants. For both we let the subjects describe tasks for a robot. We took recordings and transcribed them. Additionally, the recordings were processed by two different automatic speech recognition systems, Google’s Speech API[b] and IBM Speech to Text[c] [26]. For all three textual representations of the utterances we annotated the expected control structures and their respective reference frames to provide a gold standard. We calculate precision, recall and the F_1 score to measure the performance of our approaches. In the following subsections we first present the experimental design of our studies, afterwards we discuss the results.

6.1. ***Experimental Design***

Our user studies each comprise two scenarios containing a household robot in a kitchen setting. In all scenarios the robot is supposed to fulfill a certain task. The tasks were designed to provoke the use of conditionals, loops, and concurrency. Subjects were encouraged to describe the necessary steps to accomplish the tasks. The instructions the participants received contained no concrete wording, just a

[b] `https://cloud.google.com/speech/`, 2017-10-04
[c] `https://www.ibm.com/watson/developercloud/speech-to-text.html`, 2017-10-04

Manual Transcript: please go to the cupboard and bring me juice
ASR Transcript: please go to the what god and bring me jewels

Fig. 13: Example of an alignment between a transcription and an ASR output.

high level description of the tasks and figures that showed the setting. We took continuous recordings, one per subject and scenario. As the subjects were able to describe the task as they liked, we received quite different recordings. They vary in length from five up to 28 seconds and in instructions for the robot from a minimal set of two up to thirteen. We transcribed all recordings according to the guideline by Kiesling et al. [27] and prepared a gold standard for each. Additionally all recordings were transcribed with Google's Speech API and IBM Speech to Text to measure the robustness of our approaches to speech recognition errors.

To compare the results for the outputs of the ASRs with the ones for the manual transcripts, we aligned the text produced by the ASRs with the transcripts as shown in Figure 13. If the ASR produces more or less words the output is aligned at the next matching word. Words that have not been recognized by the ASR can result in a false negative. Similarly, words produced by the ASR that do not occur in the transcripts may result in false positives. In average IBM's ASR produces more words than the manual transcripts have. In most cases the ASR does not recognize long words and generates unrelated alternatives instead, like in Figure 13 (*cupboard* vs. *what god*). Contrarily, Google's ASR produces less words, mainly because it does not generate an output if its confidence is not above a threshold.

To retrieve the output of our agents we first run the pre-processing pipeline of *PARSE*. Afterwards, we execute the agents for action detection and coreference resolution multiple times to simulate *PARSE*'s parallel SLU processing. The resulting graph is fed into the respective agent for evaluation.

6.1.1. *Conditionals*

The user study designed to evaluate conditional detection consists of two scenarios with the following tasks. In the first scenario the robot should take the dirty dishes from the table and put them into the dishwasher and the clean dishes into the cupboard. In the second scenario it is supposed to prepare the long drink *Screwdriver*. Therefore, the robot should go to the fridge and check if it contains fresh oranges. If that is the case it should get them together with the vodka; otherwise the robot is supposed to get the orange juice instead.

Nineteen subjects participated in this study, three female and sixteen male. Fifteen are undergraduate students from different departments, the remainder are graduates. All but two (Russian, Azeri) are native German speaker. However, all but five assessed their own English skills to be at least ‘advanced’ (*CEFR* level B2) and eight of them even higher. All participants gave descriptions for both scenarios. An overview of the collected data is shown in Table 4.

[go INDP] [to INDP] [the INDP] [fridge INDP] [if IF] [there IF] [is IF] [milk IF] [bring THEN] [it THEN] [to THEN] [me THEN] [otherwise ELSE] [bring ELSE] [me ELSE] [water ELSE]

Fig. 14: Example of the gold standard for conditionals.

Table 4: Overview of the evaluation data for the user study on conditionals.

	Scenario 1	**Scenario 2**	**Total**
Recordings	19	17	36
Words (manual transcripts)	556	538	1094
Words (Google ASR)	547	516	1063
Words (IBM ASR)	590	578	1168
Conditionals	28	19	47
Recordings without conditionals	4	2	6
Conditionals without else-clause	17	9	26

We prepared a gold standard for each transcription. Unfortunately two recordings for the second scenario were unusable. The grammatical flaws made it impossible to infer the intended script, i.e., we were not able to provide a gold standard.

We annotate the clauses as per-word labels. Thus, a word can either have the label `INDP` (independent phrases), `IF` (if-clause), `THEN` (then-clause), `ELSE` (else-clause). The gold standard for the utterance, "go to the fridge if there is milk bring it to me otherwise bring me water," is illustrated in Figure 14. This labeling approach implicates that a word labeled with `THEN` which should have been labeled with `ELSE` results in a false negative (the missing `ELSE` label) and a false positive (the `THEN` label which was not expected).

6.1.2. *Concurrency and Loop*

The two scenarios of the second user study are set-up in the same environment as the first but comprise tasks aiming at a different purpose. In the third scenario the robot should take a cup to the sink and read the news while washing the cup. In the fourth scenario the robot is supposed to empty the dishwasher. We anticipate that subjects use concurrency in the third and repetition in the fourth scenario.

Ten subjects participated in this study, two were female and eight male. Four are graduate and two are undergraduate students from different departments. The remainder are PhD students. All are native German speaker. However, seven assessed their own English skills to be at least 'experienced'(*CEFR* level C1), three 'proficient'(*CEFR* level C2). All participants gave descriptions for both scenarios. In nineteen of twenty recordings the subjects used concurrency or repetition. An overview of the collected data is shown in Table 5.

[wait $_{CS_1\text{:Act1}}$] [until $_{CS_1\text{:Key}}$] [the laundry is done $_{CS_1\text{:Cond}}$] [while $_{CS_2\text{:Key}}$]
[taking out the laundry $_{CS_2\text{:Act1}}$] [check its moisture $_{CS_2\text{:Act2}}$]

Fig. 15: Example of the gold standard for concurrency and loops.

Table 5: Overview of the evaluation data for the user study on concurrency and loops.

	Scenario 3	**Scenario 4**	**Total**
Recordings	10	10	20
Words (manual transcripts)	282	287	569
Words (Google ASR)	282	280	562
Words (IBM ASR)	299	314	613
Control Structures	10	9	19
Actions	20	17	37
Recordings without Control Structures	0	1	1

For the preparation of the gold standard for concurrency and repetition, we determine the keyphrase, the condition (where applicable), and the respective actions. The gold standard for the utterance, “wait until the laundry is done while taking out the laundry check its moisture,” is illustrated in Figure 15. The control structures are numbered consecutively (CS*); the tag is completed by the elements (Key, Cond, and Act*). Note that the number of actions varies. Thus, the number of expected elements per control structure is different. Consequently, longer control structures (i.e., the ones containing more actions) influence precision and recall stronger. However, otherwise determining the dependent clause (concurrent and loop body clauses) would be imprecise.

6.2. *Results*

To assess the validity of our approaches we measure precision, recall and F_1 score. For each approach we compare the results for manual transcripts and transcripts provided by the ASR. Additionally, we determine the influence of coreference information for each of the settings.

6.2.1. *Conditionals*

The evaluation results for the conditional detection user study are summarized in Table 6. The results are promising. On manual transcripts we achieve an F_1 score of 0.862 without coreference information. For ASR outputs the results are equally good, 0.736 for Google and 0.751 for IBM. The difference between the values for transcripts and ASR outputs is in the expected range. Both ASR have a word error rate (WER) of approximately 15%. Unfortunately, a word error does not result in

Table 6: Conditional detection evaluation results.

Option	**Precision**	**Recall**	$\mathbf{F_1}$
Transcription	0.930	0.803	0.862
Transcription with coreference	0.934	0.864	0.898
Google Speech	0.817	0.670	0.736
Google Speech with coreference	0.824	0.704	0.760
IBM Speech to Text	0.862	0.665	0.751
IBM Speech to Text with coreference	0.869	0.712	0.783

an individually missed word; they affect the surrounding words as well. E.g., if an ASR does not recognize a keyword the whole conditional is likely to be missed. Moreover, word errors lead to grammatically erroneous phrase structures, which can not be parsed with our grammars. Thus, achieving F_1 scores of at least 0.736 is a promising result. The slightly better result for IBM's ASR can be explained by its generally lower WER.

The evaluation results show that our approach focuses on precision (up to 0.930) rather than recall (up to 0.803). We identified three main causes for missing a conditional. First, the pre-processing tools produce erroneous output. Besides the word errors introduced by the ASRs, the part-of-speech and chunk taggers add noisy data as well. Thus, our approach has to deal with unexpected phrase structures. In these cases our grammars can not produce entirely correct clause structures. The same applies to the second reason: unexpected wording by the subjects. Some subjects gave grammatical wrong descriptions. In most cases, our approach was able to identify the intended conditionals partially. Finally, a few subjects used sentence structures we had not in mind when we designed the grammars. An example are noun phrases directly succeeding an adverb. In most cases these noun phrases indicate the start of a new clause. However, in rare cases the noun phrase belongs to the preceding clause. Our grammars consider the more common case only.

The approach to use coreference information to extend the reference frames improves the recall remarkably (improving the recall on transcripts by 7.6%). The results by clause type depicted in Table 7 show that only then- and else-clauses are improved. The subjects did not use any references in conditional clauses. Thus, the values remain unchanged. The positive effect on the recall for then- (14.2%) and else-clauses (14.6%) can be explained as follows. Some subjects gave rather long description with clauses composed of several phrases. Our syntactical approach is able to identify simple compounds of phrases only. The coreference approach can identify long sequences comprising several dependent instructions as long as the same entity is referred more than once. Encouragingly, also the precision increases slightly. We expected a drop in precision as our coreference approach can introduce false positives. At least for our small data set this was not the case.

Table 7: Evaluation results by clause type (Transcription with/without coreference).

Clause Type	**Precision**	**Recall**	**F_1**
conditional clause	0.960	0.932	0.946
conditional clause with coreference	0.960	0.932	0.946
then-clause	0.900	0.795	0.844
then-clause with coreference	0.911	0.908	0.910
else-clause	0.944	0.405	0.570
else-clause with coreference	0.951	0.464	0.624

Table 8: Concurrency detection evaluation results.

Option	**Precision**	**Recall**	**F_1**
Transcription	0.889	0.800	0.842
Transcription with coreference	0.750	0.800	0.774
Google Speech	0.867	0.433	0.578
Google Speech with coreference	0.867	0.433	0.578
IBM Speech to Text	0.933	0.467	0.622
IBM Speech to Text with coreference	0.933	0.467	0.622

6.2.2. *Concurrency*

The outcome of the evaluation for concurrency detection is summarized in Table 8. The results for manual transcripts are promising. The F_1 score of 0.842 shows that the approach is generally feasible. Most notably, the high precision is encouraging. The usage of secondary types for keyphrases only improved recall but does not introduce additional false positives. In total, our approach detects nine out of ten concurrent sections. In the process it only determines three actions inappropriately.

The usage of coreference information does not improve recall. As shown in Table 5 the subjects stated twenty concurrent actions. With ten concurrent sections in total, that means all actions occur in pairs. This kind of clause structure is already covered by our grammar-based approach. Thus, coreference information is not useful to discover additional actions. Instead, the coreference extension introduces five false positive actions.

The results for the ASR transcripts are less positive. Our approach only detects half of the concurrent sections. Additionally, we miss one (IBM Speech to Text) respectively two (Google Speech) actions. These numbers are the result of word recognition errors. E.g., both ASR struggle with the word *while* and recognize *why* instead. This erroneous recognition is responsible for two of the five missed concurrent sections. Just like on manual transcripts coreference information has no influence. At least, precision remains at a high level in all variants; we do not detect a false positive concurrent section and only a negligible number of actions.

Table 9: Loop detection evaluation results.

Option	**Precision**	**Recall**	$\mathbf{F_1}$
Transcription	1.000	0.514	0.679
Transcription with coreference	1.000	0.686	0.814
Google Speech	0.917	0.324	0.478
Google Speech with coreference	0.882	0.441	0.588
IBM Speech to Text	0.889	0.235	0.372
IBM Speech to Text with coreference	0.889	0.235	0.372

6.2.3. *Loops*

Table 9 shows the evaluation results for loop detection. The F_1 score of 0.679 for the basic approach on manual transcripts is satisfactory. However, we only detect six of the nine loops the subjects stated. Additionally, our grammar-based approach misses eleven actions, five of them are missed because the loop keyphrase was not detected. To some extent, the outcome is attributable to the language experience of the subjects. As discussed in Subsubsection 6.1.2, all are native German speaker. As a consequence, they used some unusual grammatical constructs such as, “for every piece you find in there” or “if there are still any pieces left repeat the step.” Our grammar-based extraction of keyphrases and conditional statements is unable to cope with such expressions.

The conservativeness of our approach is shown by a precision of 1.000. This is a pleasant result in our use case. However, recall is relatively low (0.514).

We address this issue by the use of coreference information. It improves the performance on manual transcripts considerably. We are able to detect six additional actions. That means, with the help of coreference information we are able to determine all actions if we successfully detect a keyphrase. Furthermore, the coreference extension does not produce any false positive results. As a result, the F_1 score increases by 20%.

Again, the results for the ASR transcripts are worse. Our approach only detects four (Google Speech) respectively three (IBM Speech to Text) of the expected loops. Thus, overall recall decreases remarkably. Once more, word recognition errors on keyphrases are the primary reason. E.g., IBM Speech to Text recognizes the phrases “do this until” as “do it is under” in one recording. Consequently, our approach misses the keyphrase *until*, the following condition, and all three actions that are supposed to be repeated. This results in five false negatives for this example alone. Since Google Speech recognizes this phrase correctly and coreferential extension is applicable in this case, this also explains the difference in the results for the ASR.

7. Conclusion and Future Work

We have presented an approach to detection of control structures in spoken utterances. It is part of the research project *PARSE*. *PARSE* aims at end-user programming in natural language. In this context, the mapping of conditionals, repetitions, and concurrent actions in natural language to their programmatic counterparts is a crucial task. Fortunately, programmatic constructs such as if-then-else, do-while loops, and parallel sections are derived from the respective language concepts. However, natural language is less formal. Thus, the mapping from language to programmatic concepts is not trivial. E.g., the extent of dependent clauses in natural language is uncertain while blocks are stated explicitly in programming languages. Also, keywords can be omitted in natural language. Therefore, a simple keyword-based approach is infeasible.

We have implemented three so-called *SLU agents*. The agent that detects conditionals uses part-of-speech and chunk tags to identify if-, then-, and else-structures. We apply context-free grammars and heuristics that rely on coreference information to infer the reference frame of the conditionals. The agents that detect loops and concurrent sections rely on a SRL-based action detection provided by another agent. The approaches are detached from each other and further linguistic analyses. Therefore, we are able to evaluate them intrinsically. First results are encouraging. We have conducted two user studies with four scenarios comprising a household robot in a kitchen setting; 29 subjects participated in total. Our conditional detector achieved an F_1 score of 0.898 on manual transcripts inputs and 0.783 on outputs generated by IBM's automatic speech recognition system. The loop and concurrency approaches are comparable on manual transcripts (F_1 0.814 and 0.842), but more sensitive to word recognition errors. Furthermore, we have shown that the usage of coreference information improves results remarkably (up to 9.5% on F_1 score for else-clauses and up to 20% for loop body clauses).

Future work will focus on a more precise inference of the dependent clauses, i.e., the reference frame. The context model we have proposed in [19] might be useful. For the time being, we had to resort to a rule-based approach because of data sparseness. However, the feature set we use is well suited to be applied to statistical methods. We plan to implement and compare various machine learning approaches as soon as we have a sufficient data set available. Then, our rule-based approaches can serve as strong base lines.

References

[1] J. R. Bellegarda, Spoken Language Understanding for Natural Interaction: The Siri Experience, in *Natural Interaction with Robots, Knowbots and Smartphones* (Springer, New York, NY, 2014), pp. 3–14.

[2] R. Dale, The return of the chatbots, *Natural Language Engineering* **22** 811–817 (September 2016).

[3] W. R. Cook, Applescript, in *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages HOPL III*, ACM, New York, NY, USA, pp.

1–1–1–21.

[4] D. Price, E. Riloff, J. Zachary and B. Harvey, NaturalJava: A Natural Language Interface for Programming in Java, in *Proceedings of the 5th International Conference on Intelligent User Interfaces IUI '00*, ACM, New Orleans, Louisiana, USA, pp. 207–211.

[5] A. Begel, Spoken Language Support for Software Development, in *2004 IEEE Symposium on Visual Languages and Human Centric Computing* pp. 271–272.

[6] A. Begel and S. Graham, Spoken programs, in *2005 IEEE Symposium on Visual Languages and Human-Centric Computing* pp. 99–106.

[7] V. Le, S. Gulwani and Z. Su, Smartsynth: Synthesizing Smartphone Automation Scripts from Natural Language, in *MobSys'13* **2**, p. 5.

[8] C. Quirk, R. J. Mooney and M. Galley, Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* pp. 878–888.

[9] I. Beltagy and C. Quirk, Improved semantic parsers for if-then statements, in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* **1**, pp. 726–736.

[10] D. Vadas and J. R. Curran, Programming With Unrestricted Natural Language, in *Proceedings of the Australasian Language Technology Workshop* pp. 191–199.

[11] M. Steedman, *The Syntactic Process* (MIT Press, 2000).

[12] R. M. Kaplan and J. Bresnan, Lexical-functional grammar: A formal system for grammatical representation, *Formal Issues in Lexical-Functional Grammar* 29–130 (1982).

[13] C. Pollard and I. A. Sag, *Head-Driven Phrase Structure Grammar* (University of Chicago Press, August 1994).

[14] M. Landhäußer and R. Hug, Text Understanding for Programming in Natural Language: Control Structures, in *Proceedings of the 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*

[15] H. Liu and H. Lieberman, Metafor: Visualizing Stories As Code, in *Proceedings of the 10th International Conference on Intelligent User Interfaces IUI '05*, ACM, New York, NY, USA, pp. 305–307.

[16] R. Mihalcea, H. Liu and H. Lieberman, NLP (Natural Language Processing) for NLP (Natural Language Programming), in *Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Text Processing CICLing'06*, Springer-Verlag, Berlin, Heidelberg, pp. 319–330.

[17] S. Weigelt and W. F. Tichy, Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language, in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)* **2**, pp. 819–820.

[18] G. Tur and R. De Mori, *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech* (Wiley, April 2011).

[19] S. Weigelt, T. Hey and W. F. Tichy, Context Model Acquisition from Spoken Utterances, in *The 29th International Conference on Software Engineering & Knowledge Engineering,* pp. 201–206.

[20] S. Weigelt, T. Hey and M. Landhäußer, Integrating a Dialog Component into a Framework for Spoken Language Understanding, in *RAISE'18 :IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*

[21] M. Landhäußer, S. Weigelt and W. F. Tichy, NLCI: A Natural Language Command Interpreter, *Automated Software Engineering* (August 2016).

[22] R. Declerck and S. Reed, *Conditionals: A Comprehensive Empirical Analysis* (Walter

de Gruyter, January 2001).

[23] L. Haegeman, Conditional Clauses: External and Internal Syntax, *Mind & Language* **18** 317–339 (September 2003).

[24] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa, Natural Language Processing (Almost) from Scratch, *J. Mach. Learn. Res.* **12** 2493–2537 (November 2011).

[25] X. Carreras and L. Màrquez, Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling, in *Proceedings of the Ninth Conference on Computational Natural Language Learning CONLL '05*, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 152–164.

[26] G. Saon, T. Sercu, S. Rennie and H.-K. J. Kuo, The IBM 2016 English Conversational Telephone Speech Recognition System, in *Interspeech 2016* pp. 7–11.

[27] S. Kiesling, L. Dilley and W. D. Raymond, The variation in conversation (ViC) project: Creation of the Buckeye Corpus of Conversational Speech, *Ohio State University, Columbus, OH* (2006).