International Journal of Pattern Recognition and Artificial Intelligence
Vol. 15, No. 2 (2001) 359–374

UNDERSTANDING ARITHMETIC PROBLEMS IN TURKISH*

A. C. CEM SAY

Department of Computer Engineering, Boğaziçi University,
Bebek 80815, Istanbul, Turkey

This paper describes ALİ, the first computer program which can solve primary school-level arithmetic problems stated in Turkish. This task involves several subtasks: morphological analysis of every word in the input, syntactic analysis of each sentence, semantic analysis of words, providing a description of the commonsense world that is large enough to enable the correct answering of the problems, and generating the answer in human-readable form.

Keywords: Natural language processing; artificial intelligence; Turkish; commonsense; syntactic parsing.

1. Introduction

In the early years of NLP research, a number of computer programs[4,5,12] whose aim was to solve mathematics or physics problems stated in English were developed. These programs embodied a variety of approaches, ranging from relatively simple pattern-matching techniques[4] to ones involving more powerful parsing modules and a quite detailed internal representation of the rules[5,12] governing the aspects of the world relevant to the target problems' subject matter.

Building a "problem-solver" in this sense is a challenging task for many reasons. One has to deal with all levels of natural language analysis: morphology, syntax, and semantics. Since the answer to the problem has to be communicated to the user in an easily comprehensible fashion, an (at least rudimentary) capability of natural language generation also has to be attained. And, of course, the program must be "intelligent" enough to be actually able to understand and to solve the input problems; a reasonably wide portion of world knowledge has to be represented and methods of obtaining new knowledge from old have to be coded.

We have built such a program, ALİ, for solving primary school-level mathematics problems. Our program differs from the ones referred to above in the type of domain knowledge that has to be handled, in the care that must be given to morphological analysis (which is virtually nonexistent in others), and in the fact that it deals with

*The programs mentioned in this paper are available to interested researchers.
Contact E-mail: say@boun.edu.tr

the Turkish language.

The rest of this paper, which is an enhanced version of Ref. 16, is structured as follows: Sec. 2 is a brief look at previous representative work in the area of building such "problem-solving" programs. In Sec. 3, the ALİ program is described stage by stage, and illustrated by examples. Section 4 is a discussion and the conclusion.

2. Previous Work

2.1. *STUDENT*

Bobrow's STUDENT[4] was the first program for solving algebra "story problems." Coming as it does only a few years after the development of the conversation program ELIZA,[18] STUDENT employs a pattern matching technique which is similar in origins to (albeit more advanced than) ELIZA. (As a "side benefit" of our examination of these programs, we built the first "Turkish-speaking" conversation program; see Ref. 2.) STUDENT breaks up each sentence to its underlying "kernel sentences" by a process guided by the existence of certain keywords in the input, and the use of synonyms is made both to identify seemingly different phrases that refer to the same thing, and to find the operators in matching the sentence's form with that of an equation which can then be solved using the program's knowledge of algebra. Essentially, the input sentence itself is converted to an equation through a series of rewrites. When necessary, "background knowledge" about certain familiar concepts, like

```
(DISTANCE EQUALS GAS CONSUMPTION TIMES NUMBER OF GALLONS OF GAS
USED)
```

from the program's library can be used in addition to the facts in the input problem during the computation of the answer from the givens.

When stuck, STUDENT asks the user for help in determining possible relationships among the concepts represented by the phrases that it has identified as variables. Here is an example output of STUDENT (taken entirely from Ref. 4):

```
(THE PROBLEM TO BE SOLVED IS)
(TOM HAS TWICE AS MANY FISH AS MARY HAS GUPPIES . IF MARY HAS 3
GUPPIES , WHAT IS THE NUMBER OF FISH TOM HAS Q.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL G02520 (NUMBER OF FISH TOM (HAS / VERB)))
(EQUAL (NUMBER OF GUPPIES (MARY / PERSON) (HAS / VERB)) 3)
(EQUAL (NUMBER OF FISH TOM (HAS / VERB)) (TIMES 2 (NUMBER OF GUPPIES
(MARY / PERSON) (HAS /VERB))))
(THE NUMBER OF FISH TOM HAS IS 6)
```

Note that, at the level of individual words, the program is using an extremely small amount of knowledge. It has not realized that Tom is a person, and it has no idea what, for instance, guppies are. Still, it is able to give the correct answer for

this question.

STUDENT is limited also in its syntactic and semantic capabilities (the entire program is given as an appendix of Bobrow's paper!) and its underlying algorithm does not seem easily extendible to handle any problem domain other than this kind of algebra.

2.2. *ISAAC*

Novak's ISAAC[12] is a LISP program which solves problems in the domain of rigid body statics. Unlike STUDENT, ISAAC performs a "genuine" syntactic and semantic analysis of the input text sentence by sentence. The ATN-like parser makes use of semantic information to help it in its decisions. The semantic processor uses background information about "canonical objects" in its repertory (like LEVER, WEIGHT, SPRING, PIVOT, etc.) to flesh out the parts of the problem situation not explained explicitly in the text. A program named EUCLID is called for any necessary geometric computations and the construction of a geometric model of the situation. This model is utilized to write the equations which stem from the program's knowledge of the laws of physics and which apply among the objects in this particular situation. These equations are solved by a symbolic manipulation package for the "desired unknowns" which have previously been identified by the semantic analyzer, and the answer is printed out in the form of one or more noun phrases. To illustrate its "deep" understanding of the problem, the program also draws a diagram of the problem situation as part of its output. Here is a sample input/output pair, with the drawing omitted[12]:

```
(A STEEL BEAM OF UNIFORM CROSS SECTION WEIGHS 2.50000E+8 DYNES)
(IF IT IS 500 CM LONG, WHAT FORCE IS NEEDED TO LIFT ONE END OF IT)
ANSWER: 1.25000E+8 DYNE
```

ISAAC was effectively "built around" 20 problems taken from a few high school and college physics textbooks. The vocabulary of the program consists of about 200 words. But within this limitation, the problems of syntax, semantics, and internal world-modeling have been approached carefully and "legitimately." We adopted a similar methodology in the construction of ALİ.

2.3. *MECHO*

MECHO[5] is a collection of PROLOG programs which solves mechanics problems. (Newtonian mechanics seems to attract writers of such programs, maybe because its basic laws are simple enough to be coded easily, and yet hard enough so that the programs are considered "intelligent" by laymen.) MECHO has solved, in particular, problems about pulleys, motion on smooth complex paths, and motion under constant acceleration. The sentences in the input text are processed by the syntactic parser to produce a collection of PROLOG facts representing the question. The semantic routines are called by the parser with specific questions like "Can

this verb relate these objects?" during the syntactic analysis to determine the correct branch in the tree of possible parses. Certain words (like "pulley") in the text cause the "cueing" of "schemata" containing background knowledge about specific problem types; this results in the creation of objects which, although not mentioned in the text, have to appear in a model of the explained kind of situation. (Note the similarity to the use of canonical objects in ISAAC.) Once the text has been translated into assertions, a general purpose equation extraction algorithm is employed to write formulae for the "sought" quantities. Prediction questions, e.g. those about the roller-coaster world (see for instance Ref. 13) which cannot be answered using qualitative information alone are transformed into ones requiring computation of one or more "soughts" and treated similarly. The equations are solved by an algebra package. Memory limitations force the user to hand-feed the results of one module to another. An example trace of a sequence of executions for one problem is presented below[5] (with a lot of messages about the proceeding of the processing omitted).

```
| Two particles of mass B and C are connected by a light string
| passing over a smooth pulley.
{omitted stuff}
yes
|
| continue.
|
| Find the acceleration of the particle of mass B.
{omitted stuff}
FINAL ANSWERS ARE
tension1=-1*(g*c+b*-1*g)*(-1*b+(-c)):-1*b+b*g &
a1=-1*(g*c+b*-1*g)*(-1*b+(-c)):-1 & true
```

Note that an additional quantity representing the tension in the string, which is introduced when the algorithm seeks an equation for the acceleration, is also computed and output.

3. ALİ

In Ref. 4, Bobrow gives the following "operational" definition of "understanding":

> "A computer *understands* a subset of English if it accepts input sentences which are members of this subset and answers questions based on information contained in the input."

In the spirit of this definition, we state that ALİ understands a subset of Turkish.

At the beginning of this project, we randomly selected 23 "pure-text" problems from a Turkish mathematics textbook[3] for third year primary school students. Apart from a few minor changes which we introduced since we felt that the original

forms of some of the questions were grammatically wrong, the program accepts the problems as they appear in the textbook, and presents the answers in a neat fashion, usually in the form of a sentence. The knowledge of mathematics required for the solution of these problems is restricted to the four arithmetic operations and basic facts about geometric figures like triangles and squares. The "commonsense" knowledge required is wide, and ranges from items like "Cows are animals" to "There are 52 weeks in a year."

ALİ is in the form of several executable Turbo-PROLOG project files, linked through an MS-DOS batch file, running on an IBM-compatible PC.

When the program starts execution, an editor screen is activated, and the user is expected to enter the problem text, obeying the orthographical rules imposed by the analyzer. When the completion of entry is signaled, the text file containing the problem is passed to the morphological parser.

We will follow the processing of a simple example throughout this section. Other examples will be presented when necessary for the explanation of the program's more advanced features. Our first example is:

```
bir yılda kaç hafta bulunur ?|
```

("How many weeks are there in a year?", or, more precisely, "How many weeks are found in a year?") (This is the first problem to be solved successfully by ALİ.)

3.1. *Morphological level*

Unlike English, Turkish has a rich morphological structure, and hence unlike the programs described in Sec. 2, ALİ has to carry out a morphological analysis of each word in the input before proceeding with the syntactic parsing. In this stage, a text tagger obtained by slightly modifying a morphological parser for Turkish originally constructed for a Turkish–Azeri machine translation project[8] is called. The usage of this general-purpose parser means that ALİ's "syntactic" vocabulary is much larger (on the order of 20 000 words) than those of ISAAC and MECHO. (The "semantic" vocabulary is comparatively limited, as explained in Sec. 3.3.)

The text tagger has some shortcomings, most of which stem from the somewhat specialized nature of the sublanguage under consideration. For example, "words" like "15'ten" ("from 15") which appear in some counting problems involve a vowel harmony rule that requires the pronunciation of the number to be known. ALİ contains a morphological postprocessor which goes through such cases where the text tagger has failed, and produces the appropriate morphological parses. For the above example, a predicate which converts numbers ("15") to the corresponding words ("on beş") is employed, and the parse representing that the root "15" has taken the ablative suffix is obtained.

The result of the morphological phase is passed to the syntax analysis phase in the form of a list of lists (one for each sentence) of lists (one for each word) of lists

```
[[<"bir",[<adjective;root(adjective("bir"))>]>,
<"yılda",[<noun;root(noun("yıl")),case(locative)>]>,
<"kaç",[<adjective;root(adjective("kaç"))>,
<verb;root(verb("kaç"))>]>,
<"hafta",[<noun;root(noun("hafta"))>]>,
<"bulunur",[<verb;root(verb("bul")),voice(passive),tense(aorist)>]>
]]
```

Fig. 1. Morphological parse of the example problem.

representing possible morphological parses.

Figure 1 shows the result of the morphological parsing stage for our example problem. As seen, all possible parses of each word are listed. The parse lists start with the word's final syntactic category (e.g. "adjective"), and go on with the root and the suffixes (if they exist). In our example, two separate parses have been found for the word "kaç," which means both "how many" and "run away." It is up to the later stages to find out which one of these is the correct meaning in this context.

3.2. *Syntactic level*

The syntax analysis phase finds, for each sentence in the problem, all distinct parses possible according to our grammar, through forced backtracking. Initially, we had planned to perform the complete (syntactic + semantic) processing of a sentence before proceeding with the next sentence, but the memory limitations of the particular implementation of PROLOG that we use rendered such an approach impossible for a program of this size. In ALİ's present form, semantic processing starts only after parsing of all sentences has been completed, and takes a file containing the results of the parser as input.

Unlike ISAAC and MECHO, the usage of semantic information is kept to a minimum during parsing. One example where we do use such knowledge is as follows: the parser checks whether the second word in a phrase like "5 metre ip" ("5 meters of rope") is "unit-like" or not. Only if the second word *is* unit-like does the parser assign the special noun phrase structure ⟨Measurement + Unit + Noun Phrase⟩ to this group of words.

The actual parsing algorithm we use is a modification of the difference list recognition framework, explained in Refs. 6 and 7. This is a top–down, left-to-right parser which is very easy to write in PROLOG. Our version for Turkish is somewhat more complicated than the one in Ref. 7, since, where Ref. 7 uses a simple list of words to keep the sentence, we employ a more complex structure, as described at the end of Sec. 3.1. The predicate which consumes terminals from that list has also been modified, so that no possible distinction arising from multiple morphological parses for a word is overlooked.

$S \rightarrow NP\ V$

$S \rightarrow NP\ NP\ V$

$S \rightarrow NP\ NP\ NP\ V$

$S \rightarrow AdvP\ AdvP\ V$

$S \rightarrow AdvP\ AdvP\ AdvP\ V$

$S \rightarrow NP\ NP\ AdvP\ V$

$S \rightarrow AdvP\ NP\ AdvP\ V$

$S \rightarrow S_{ise}\ S$ {S_{ise} is of the form "S + ise", i.e. "If S"}

Fig. 2. The S rules.

The grammar itself covers all sentences appearing in our target problem set adequately, and is a rather expressive subset of Turkish. Figure 2 contains a summary of the sentence rules in our grammar. Whether Turkish has a legitimate verb phrase structure has been under investigation by linguists.[10] For no reason other than ease of coding, we adopted the idea that the language has no verb phrase complex in this work, as can be seen in the figure. Turkish allows "free" deletion of sentence constituents, such that any one or more of the constituents may be missing from the sentence. The lack of NP in a rule may be due to this general property of the language or to the fact that the imperative constructions universally lack an overt subject. The rules of Fig. 2 involve additional checks (not indicated in the figure) about the agreement of the subject NP and the verb of the sentence in their "person" attributes. (In the right-hand sides, if there is only one nominative NounPhrase, then it is the subject of the sentence. If two nominative NP's exist, the first one is the subject.)

Since checks about, among other things, the number of objects that specific verbs can take, are not implemented, the grammar clearly overgenerates. Most (ideally, all but one) of the parses for each sentence will be simply "wasted" by the semantic processor, since semantic routines corresponding to them will not be found.

Let us continue following our example problem. Since the problem contains a single sentence, the list in Fig. 1 has a single element, namely, the list containing all parses of all the words. Since the word "kaç" has two possible parses, we obtain two candidate sequences for this sentence (Fig. 3). These sequences are checked for compliance with our grammar by attempting to match them with the rules, some of which are shown in Fig. 2. The PROLOG clause in Fig. 4 is the implementation of the second rule in Fig. 2. As a result of the syntactic analysis stage, the correct syntactic parse of our sentence will be found to match the particular rule described in Fig. 4, and that parse (Fig. 5) will be passed to the semantic stage for further processing.

```
[<"bir",<adjective;root(adjective("bir"))>>,
  <"yılda",<noun;root(noun("yıl")),case(locative)>>,
  <"kaç",<adjective;root(adjective("kaç"))>>,
  <"hafta",<noun;root(noun("hafta"))>>,
  <"bulunur",<verb;root(verb("bul")),voice(passive),tense(aorist)>>
  ]
[<"bir",<adjective;root(adjective("bir"))>>,
  <"yılda",<noun;root(noun("yıl")),case(locative)>>,
  <"kaç",<verb;root(verb("kaç"))>>,
  <"hafta",<noun;root(noun("hafta"))>>,
  <"bulunur",<verb;root(verb("bul")),voice(passive),tense(aorist)>>
  ]
```

Fig. 3. Two candidate sequences corresponding to the example sentence.

```
s(s([np(NP1,loc,Ag1,Poss1),np(NP2,nom,AgrSubj,Poss2),
  vp(V,Tense,CTense,Sense,AgrVerb,Voice)]),Input,Rest):-
  np(np(NP1,loc,Ag1,Poss1),Input,P1),
  np(np(NP2,nom,AgrSubj,Poss2),P1,P2),
  vp(vp(V,Tense,CTense,Sense,AgrVerb,Voice),P2,Rest),
  agree(AgrSubj,AgrVerb).
```

Fig. 4. A sentence rule in PROLOG.

```
pa(1,s([np([adj([adj("bir")])],np([noun([noun("yıl")])],loc,sg3,
  poss(frpos))],loc,sg3,poss(frpos)),np([adj([adj("kaç")])],
  np([noun([noun("hafta")])],nom,sg3,poss(frpos))],nom,sg3,
  poss(frpos)),vp([verb([verb("bul")],frcs,frpos,frmo,aorist,
  frct,positive,sg3,passive)],aorist,frct,positive,sg3,passive)])).
```

Fig. 5. Syntactic parse of the example sentence.

3.3. *Semantic level*

The semantic processor's input is the problem text and all parses of each of the sentences in the problem. It considers each sentence S to see if S has a parse which matches one of the semantic "templates" that are defined in its code. If a parse successfully matches a semantic template, a corresponding action is performed by the computer. This "action" may be the output of some information (for question or command sentences) or simply a modification of the internal world model of the program.

The templates describe simple sentence forms (i.e. with no subordinate clauses). Presently the only rule in our grammar that allows sentences with subordinate clauses is

$$\mathrm{S} \rightarrow \mathrm{S}_{\mathrm{ise}}\mathrm{S}\,.$$

Sentences with parses of this form, like "97 simitin 68 tanesi satılırsa kaç simit kalır?" ("How many simits (a Turkish bakery product) are left if 68 out of 97 simits are sold?") are treated by the semantic processor as two consecutive "nonconditional" simple sentences. For instance, the above-mentioned sentence is handled as "97 simitin 68 tanesi satılır. Kaç simit kalır?" ("68 out of 97 simits are sold. How many simits are left?") and the two new sentences are matched separately to different templates. This treatment of the conditional is adequate for all texts in our target set; for a general purpose application, this approach would certainly have severe weaknesses.

The semantic templates have been prepared after a careful examination and classification of the "types" of sentences that appear in our target set of problems. The categories we identified are as follows:

1. Declarations about the existence of a certain amount of Things in a certain Location.
   Example: "Bir fabrikada 217 işçi vardı." ("There were 217 workers in a factory.")
2. Declarations about changes (increases or decreases) in the amounts of previously mentioned Things.
   Example: "Sonra 65 öğrenci daha geldi." ("Later, 65 more students arrived.")
3. Commands about counting in a certain range.
   Example: "36'dan sıfıra kadar dörder dörder sayınız." ("Count down from 36 to zero in fours.")
4. Questions (or "find" commands) about the amounts of Things. (These may require arithmetic operations and commonsense knowledge to compute. The word "amount" is used very loosely here, and the length of, say, the perimeter of a geometric figure may also be sought in this regard.
   Example: "Geriye kaç kilogram havuç kalır?" ("How many kilograms of carrots are left?")
5. "Multiplication" questions about the amount of Things in k (an integer) Locations. (The amount of Things in one such Location must be already stated or otherwise known.)
   Example: "9 perde kaç metre bezden yapılır?" ("How many meters of cloth are needed to make 9 curtains?")
   (Note that our main example "How many weeks are there in a year?" is a very simple specimen of this type.)
6. Declarations about "linear" relationships between two quantities linked by an action.
   Example: "Bir bakkal günde 28 şişe süt satıyor." ("A grocer sells 28 bottles of milk in a day.")
7. "Multiplication" questions about the kind of linear relationships mentioned in (6).
   Example: "15 günde kaç şişe süt satıyor?" ("How many bottles of milk does he/she sell in 15 days?")

The basic idea is to have one such template which will handle all the (possibly syntactically differing) sentence parses for each of these semantic categories. Parses with verbs which have the same semantic effect for our purposes are rearranged by a semantic preprocessor before a match with the appropriate template is attempted. For example, although the verbs "çıkmak" ("to get out"), "ölmek" ("to die"), "emekli olmak" ("to go to retirement"), "satılmak" ("to be sold"), and indeed, "satmak" ("to sell") all indicate a decrease-like (type 2) meaning, the noun phrase whose referent is the entities that are decreasing appears in the subject position for "to get out", but in the direct object position for "to sell". The preprocessor notices this and passes the constituents to the semantic template in a standard order.

Once a template is matched, the computer performs certain checks and operations on the constituents (usually noun phrases) of the parse at hand. For example, a parse of the form $\mathrm{NP}_{\mathrm{accusative}}\ \mathrm{V}(\text{"bul"})_{\mathrm{imperative,\ 2^{nd}person}}$ ("Find" NP) will match a template of type 4, and cause a predicate named `compute_np`, whose function is, as its name suggests, to compute a number, a unit, and, if possible, a "type name" describing the referent of the NP. Possible results of this predicate are tuples like ⟨5, m, kumaş⟩ ("5 m of cloth"), ⟨2, tane, elma⟩ ("2 apples"), ⟨4,cm⟩. "Computing" a noun phrase may involve resolving pronouns, making use of commonsense knowledge, and arithmetic operations. See Secs. 3.4 and 3.6 for further details of, and examples about, this process. Section 3.5 will briefly explain how natural language answers are generated by the templates for questions and commands. The semantic processor terminates successfully if it has been able to match a template to each of the sentences in the input text.

It has to be stressed that the very specialized subject domain makes such a use of templates feasible. Of course, for general purpose text understanding, this approach would be impractical, to say the least, and a much more advanced scheme, possibly in the lines of Schank's CD theory[17] would be needed.

As for our main example, we have already stated that this sentence matches template type 5. The semantic processor searches for two noun phrases (one locative and one nominative) in the sentence. The syntactic parsing result seen in Fig. 5 satisfies these conditions. The locative noun phrase, that is,

```
np([adj([adj("bir")]),np([noun([noun("yıl")])],loc,sg3,poss(frpos))],
loc,sg3,poss(frpos))
```

shows that the Location in which the Things whose amount is being asked will be sought is one ("bir") year ("yıl"), and the nominative noun phrase, that is,

```
np([adj([adj("kaç")]),np([noun([noun("hafta")])],nom,sg3,
poss(frpos))],nom,sg3,poss(frpos))
```

shows that the Things whose amount is being asked are weeks ("hafta").

3.4. *World knowledge*

It has been known for a long time that any reasonably general purpose AI program must have access to a huge amount of commonsense knowledge about the world. This knowledge is what the authors of mathematics problems, novels or encyclopedia entries assume that their readers already know, and hence do not explicitly mention. Research about how best to represent this information is a "hot topic" of AI, and has been going on in earnest since the 1970's.[9] Lenat's CYC[11] is currently the closest to a general purpose commonsense-base, and one of the aims of such programs is, in fact, to serve as the commonsense checker routine to NLP programs like ALİ. In the absence of the chance of easily incorporating such a comprehensive engine to ALİ, we went on to design our own representation of the world that the program has to possess in order to solve the target problems successfully. Here is a taste of what this model contains:

We have an "isa" hierarchy[14] which is used mainly for identifying referents of noun phrases. For example, in the semantic processing of the last sentence of the problem

"Bir çiftlikte 255 koyun, 67 kuzu, 8 inek vardır. Bunların hepsi kaç hayvan eder?" ("There are 255 sheep, 67 lambs, 8 cows on a farm. How many heads of animals are there on this farm?") the template of type 4 which matches that parse tries to satisfy the predicate `compute_np` for the noun phrase "bunların hepsi" ("all of these"), with the aid of additional information from the other noun phrase in the question, "kaç hayvan" ("how many animals"). The pronoun resolution routine makes use of the facts

`isa(inek,hayvan)` {a cow is an animal} `isa(kuzu,hayvan)`, {a lamb is an animal} etc. in the commonsense database.

Commonsense information can be "learned" during processing as well. For instance, consider the problem

"Bir kamyonun deposunda 67 lt mazot vardı. Şoför 145 lt daha aldı. Kamyondaki mazotun hepsi kaç litre olur?" ("There were 67 lt of diesel oil in the tank of a truck. The driver bought another 145 lt. What is the total amount of diesel oil (in liters) in the truck?")
Processing of the first sentence adds, in effect, the following facts to the dynamic context maintained by the program:

`haspart (kamyon,depo)` {"tank" is a part of "truck"}
`containment(depo,[[(67,lt,mazot)]],past)` {"tank" contained 67 lt of "diesel oil" in the past}

When the words "kamyondaki mazotun" ("of the diesel oil in the truck") are met during the processing of the last sentence, a rule saying that "if D is a part

of K, then K contains whatever D contains" is employed to come up with the correct answer.

Stock information items like "1 hour contains 60 minutes", written in the format of the dynamic context, are loaded from a file at the beginning of the program, to handle such "pure knowledge" questions among the target problems. Our main example's processing makes use of such an item, namely "1 year contains 52 weeks". This fact is found after a search with the index value for "hafta" ("week"), and the answer number is calculated to be 1*52 = 52.

Needless to say, this commonsense base lacks in both breadth and width, and has to be updated seriously if it is to be used for any wider application.

3.5. *Answer generation*

As mentioned above, question or command sentences require the production of screen output by ALİ. For the answer to be intelligible, the best strategy is to give it in the form of a sentence. We use a simplistic but adequate method of language generation for this purpose.

All of the problems under consideration require one of two forms of answer. They are either counting commands, which necessitate the display of a sequence of integers on the screen, or "calculation" problems requiring a single number as the result. The template for counting problems has its special purpose answer generator, which simply prints out the appropriate sequence. For example, the problem

"15'ten geriye doğru üçer üçer sayınız." ("Count backward in threes, starting with 15.") gets the answer

```
15
12
9
6
3
0
```

The generator employed by all the other templates works as follows: if the verb of the sentence is in imperative form ("bulunuz" = "find"), then the generator takes the number computed by the previous routines in the template and the unit quantified by this number and prints them out.
Example:
"Kenar uzunlukları 3, 4, 5 cm olan bir üçgenin çevre uzunluğunu bulunuz." ("Find the perimeter of a triangle the lengths of whose sides are 3, 4, 5 cm.")

```
Yanıt: {Answer}
12 cm
```

If the verb is not imperative, a technique that we call "kaç-replacement" is used:

the word "kaç" ("how many") in the sentence is replaced by the computed number, and this sentence is printed out. For question sentences in which no word has first or second person suffixes, as is the case with all of our target problems, this method causes no syntactic and semantic anomalies, and produces a natural form of answer. Example:

"2615 civcivin 73 tanesi ölürse geriye kaç civciv kalır?" ("How many chicks remain if 73 out of 2615 chicks die?")
`2615 civcivin 73 tanesi ölürse geriye 2542 civciv kalır.` ("2542 chicks remain if 73 out of 2615 chicks die.")

And the processing or our main example results in
`bir yılda 52 hafta bulunur.` ("There are 52 weeks in a year.")

3.6. *Further examples*

Here are some more problems that ALİ has solved, presented in the program's own input/output format:

```
----------
```

```
36 kg elmanın dörtte biri kaç kilogram eder ?|
```

("What is the weight (in kilograms) of one fourth of 36 kg of apples?")

{Semantic template used: Type 4}

```
36 kg elmanın dörtte biri 9 kilogram eder.
```
("The weight of one fourth of 36 kg of apples is 9 kilograms.")
Note that the program is able to compute "verbal fractions" like "dörtte bir" ("one fourth"), and knows that "kg" and "kilogram" are the same thing.

```
----------
bir fabrikada 217 işçi vardı . 15 işçi çıktı . 7 işçi emekli oldu .
fabrikada kaç işçi kalır ?|
```

("There were 217 workers in a factory. 15 workers left. 7 workers retired. How many workers remain in the factory?")
{Semantic templates used: Type 1, Type 2, Type 2, Type 4}

```
fabrikada 195 işçi kalır.
```

("195 workers remain in the factory.")
The nonstandard spelling requirements can be eliminated easily through a preprocessor.

```
----------
çevre uzunluğu 40 cm olan bir karenin bir kenarının uzunluğu kaç cm
olur ?|
```

("What is the length of one side of a square whose perimeter is 40 cm?")
{Semantic template used: Type 4}

```
çevre uzunluğu 40 cm olan bir karenin bir kenarının uzunluğu 10 cm
olur.
```
("The length of one side of a square whose perimeter is 40 cm is 10 cm.")

```
----------
bir kasabada cumhuriyet bayramı törenine ilkokuldan 186 öğrenci ,
ortaokuldan 49 öğrenci katıldı . bunların hepsi kaç kişi olur ?|
```

("In a town, 186 students from the primary school and 49 students from the secondary school participated in the celebrations of Republic Day. What is the total number (in persons) of these?")

```
bunların hepsi 235 kişi olur.
```

("The total of these is 235 persons.") See Ref. 15 for other examples.

4. Conclusion

This paper has presented ALİ ("Aritmetikçi-Lisan İşleyici" = "Arithmetician-Language Processor"), the first (and, in the time of writing this manuscript, the most advanced) "problem-solver" computer program for the Turkish language. The various stages of the program have been developed quite independently from each other and can be replaced by programs of similar functionality from other sources with relatively little effort. So ALİ can be used as a test-bed for the performance of competing algorithms and representations for morphological or syntactic parsing, semantic representation, etc.

Our work on ALİ will continue on many fronts. Employing other parsing algorithms for better handling of Turkish syntax, which differs from Indo-European languages in its almost free word order, among other things, is on our agenda. The semantics processor currently has a somewhat shallow representation of concepts, where the actual Turkish strings like "öğrenci" ("student") are used in, say, isa facts in the model. This has the consequence that, for a synonym (e.g. "talebe" = "student") of an internally modeled word to be handled equivalently by the program, duplicate model fragments containing *its* surface form have to be added. We plan to introduce a better solution to this problem, making use of an online synonyms–antonyms dictionary[1] that we developed separately.

Acknowledgments

Thanks are due to my colleagues İlker Hamzaoğlu, Levent Akın and Sumru Özsoy for their contributions to various stages of this project. ALİ was partially supported by Grant no. 93A0152 of the Boğaziçi University Research Fund.

References

1. K. Arıcan and T. Yemliha, "Synonyms and antonyms dictionary for Turkish," B. S. Thesis, Department of Computer Engineering, Boğaziçi University, 1993.
2. Ç. Aytekin, A. C. C. Say and E. Akçok, "ELIZA speaks Turkish: a conversation program for an agglutinative language," *Third Turkish Symp. Artificial Intelligence and Neural Networks*, Ankara, 1994, p. 435.
3. M. Barış, M. K. Sümer, M. Zeytin and T. Yücel, *İlk Öğretim Matematik 3*, Ders Kitapları A. Ş., İstanbul, 1992.
4. D. Bobrow, "Natural language input for a computer problem-solving system," *Semantic Information Processing*, ed. M. Minsky, MIT Press, Cambridge, MA, 1968, pp. 135–215.
5. A. Bundy, L. Byrd, G. Luger, C. Mellish, R. Milne and M. Palmer, "MECHO: a program to solve mechanics problems," Working Paper 50, Department of Artificial Intelligence, Edinburgh University, 1979.
6. A. Gal, G. Lapalme, P. Saint-Dizier and H. Somers, *Prolog for Natural Language Processing*, John Wiley, Chichester, 1991.
7. G. Gazdar and C. Mellish, *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*, Addison-Wesley, Wokingham, 1989.
8. İ. Hamzaoğlu, "Machine translation from Turkish to other Turkic languages and an implementation for the Azeri language," M. S. Thesis, Department of Computer Engineering, Boğaziçi University, 1993.
9. J. R. Hobbs and R. C. Moore (eds.), *Formal Theories of the Commonsense World*, Ablex, Norwood, NJ, 1985.
10. J. Kornfilt, "On configurationality in Turkish," *Proc. 5th Int. Conf. Turkish Linguistics*, School of Oriental and African Studies, London, 1990.
11. D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt and M. Shepherd, "CYC: toward programs with common sense," *Commun. ACM* **33** (1990) 30–49.
12. G. Novak, "Computer understanding of physics problems stated in natural language," Ph.D. Thesis, Department of Computer Science, University of Texas at Austin, 1976.
13. M. Pakkan and A. C. C. Say, "Qualitative collision detection in the roller coaster world," *Proc. Ninth Turkish Symp. Computer and Information Sciences*, Antalya, 1994, pp. 583–590.
14. E. Rich and K. Knight, *Artificial Intelligence*, 2nd edition, McGraw Hill, NY, 1991.
15. A. C. C. Say, "Bilgisayarla Türkçe anlama," *Galatasaray Üniversitesi Mühendislik Bilimleri Dergisi* **1** (1998) 37–53.
16. A. C. C. Say, H. L. Akın and A. S. Özsoy, "A program which solves arithmetic problems in Turkish," *Proc. Ninth Turkish Symp. Computer and Information Sciences*, Antalya, 1994, pp. 550–557.
17. R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum, Hillsdale, NJ, 1977.
18. J. Weizenbaum, "ELIZA: A computer program for the study of natural language communication between man and machine," *Commun. ACM* **9** (1966) 36–44.

A. C. Cem Say received the B.S. and Ph.D. degrees, both in computer engineering, from the Department of Computer Engineering, Boğaziçi University, Istanbul, Turkey, in 1987 and 1992, respectively. He has been a faculty member in the same department since 1992. His past employers include the Turkish Naval Inventory Control Center.

His research interests are natural language understanding, qualitative reasoning algorithms, and qualitative representations.