BussTUC-manual på Wiki-format

= TUC - The Understanding Computer =

= TUC USER MANUAL =

== Version 22.1 Date 140910 ==

Revised for Confluence by Rune Sætre
Based on Version 21.4 Date 060515,
From www.idi.ntnu.no/~tagore

== VERSION MANAGEMENT ==

To ensure a disciplined distribution of the latest versions, the system should not be redistributed to third parties.
For a description of the [[#Version Management Policy]], see the end of this manual.

== MAIN DESCRIPTION ==

TUC is an acronym for The Understanding Computer.

TUC is a prototypical Natural Language Processor written in Prolog.
It is designed to be a general purpose easily adaptable natural language processor.

I consists of a general grammar for a subset of the language, a semantic knowledge base, and modules for interfaces to other systems like UNIX, SQL-databases and Traffic Information Systems.

There is at the moment two versions, one for English and one for Norwegian. We will let the English version be used for a generic description. (A suffix _e distinguishes the English version from the Norwegian version _n).

A feature of the system is to detect which language is actually typed, and then process and answer the query in this language.

The interface modules are not included in this package.

=== System Requirements ===

The implementation language is SICStus Prolog 4 (or later) which is a standard Prolog compiler.

The features of the Prolog system needed are:

- A Prolog compiler for efficiency.
- A full DCG preprocessor.
- A standard module system

=== How to get hold of the system ===

TUC is available on SVN: http://basar.idi.ntnu.no/svn/busstuc
which is at the Department of Computer and Information Science at the University of Trondheim (NTNU), Norway.

When delivered, it will expands to the bustuc/ folder containing a collection of Prolog-files.

=== How to create the system ===

NOTE the bus system contains a large database busroute.pl.

It is therefore possible to compile this separately into a saved state
(e.g. busbase) and use busbase instead of the compiler command.

Example:

(Step 1, only one time whenever the schedule is updated)

```
% sicstus
?-compile(busroute).
?-save_program(busbase).
?-halt.
.....
```

(Step 2... To compile the rest of the program)

```
% busbase.sav
?-[tucbus].
?-save_program(bustuc).
?-halt.
......
```

1. Assume that all the files are collected in a directory.
Then call the Prolog compiler to compile TUC by the
commands

?-[tucbus]. (Bus system, English version)

?-[tucbuss]. (Bus system, Norwegian version)

?-[tucunix]. (for UNIX Sicstus version)

?-[tucswi]. (for WINDOWS SWI-Prolog version)

?-[tuclinuks]. (Linux, Norwegian installation language and user language)

?-[tucbuster]. (Dialog version of BusTUC)

3. Make an an executable program , e.g. NRL (or bustuc)

?-save_program(nrl).

=== How to run the system ===

1. Load the compiled system

% nrl.sav

yes
| ?-

2. Call the system.

| ?- run.

E:

3. Tell and ask.

=== Sample session ===

The user is prompted by an E: for each new sentence.

For this example to work, it may is necessary to set these flags

?- queryflag := false.
?- busflag :=false.

depending on default settings. Otherwise, statements are taken as implicit questions.

****

E: every man that lives loves mary.

.....
(A isa man,live/A/B)=>mary isa woman
(A isa man,live/A/B)=>love/A/mary/(s1/A)
.....

E: john lives.

.....
john isa man
live/john/s3
.....

E: who loves mary ?

.....
which(A)::(mary isa woman,love/A/mary/B,A isa person)
.....

john

=== How the system works ===

The system translates the English text via a general grammar with semantic constraints.
The constraints are determined by the content of the semantic knowledge base.
The system operates on a backtracking fashion, returning solutions which are both syntactically and semantically correct.
In this version, only the first possible solution is presented, the other are cut away.

Example:

" John saw a man in the park with a telescope"

This sentence is genuinely ambiguous, but the analysis chosen will at least be semantically plausible (with the current semantic definitions).

TUC translates English to First Order Logic (FOL), and then to a Skolemized form called TQL. Slash (/) is used as a general predicate generating operator, which is left-associative. The last argument is usually
a situation-variable or -constant. In most cases, it can be understood as an unspecified time period.

Statements are translated to stored facts and rules.

Queries are processed and answered according to the knowledge base and the dialogue content.

Except for system commands, commands are meant to be performed by TUC.

A script of sentences can be read from a text file, given by a read command.
All definitions given herein will be semi-permanent.
Otherwise, definitions given in dialogue will be forgotten by reset commands.

There are a few error messages. They are accompanied by a rephrase of the input, and a '*' pointing at the error, (usually the word before or after the '*'). The '*' signifies how far the analysis proceeded until no more alternatives were possible.

The error messages are:

--- Ungrammatical at * ---

The phrase was ungrammatical, even if the semantic check was off.

--- Meaningless at * ---

The phrase was grammatical, but violated a semantic constraint

--- Incomprehensible at * ---

General error, normally followed by a list of unknown words

--- Incomprehensible words: < words >

The phrase was ungrammatical, but incomplete (e.g. no verbs)

--- Please use a complete sentence ---

If you get an error message which is not an unknown word, you should experiment with simpler versions of the phrase.
Hopefully, missing semantic definitions should be the main source of the problem.

For the description below, the user types whatever is not printed. Text surrounded by < > is generic and is not verbatim.

=== System Commands in Prolog mode: ===

?-tuc. Initiating call. Calls start

?-start. Prints version data, starts

?-restart. Erases temporary and semi-permanent memory and starts

?-erase. Erases semi-permanent memory

?-reset. Erases temporary memory.

?-clear. Resets trace and debugs.

?-run. Normal go

?-hi. Calls NLP with debug on.

?-ho. Calls NLP with debug off.

?-makegram. Compile a new grammar.

This command must be given after ?-[gram_e].

?-dialog. Sets system in dialog mode. (Experimental)

?-status. List the contents of dialogue memory.

?-testgram. Set spy-points on the grammar for debugging.

?-set(<parameter>,<value>). Dynamically sets parameter to value

?- X := Y. Same as ?-set(X,Y).
X and Y must be constants

?- X =: Y. Same as value(Y,X).

?- value(P) Prints the current value of the parameter P.

?- readscript. Reads a file for permanent store.

?- norsk. Change language to Norwegian.

?- english. Change language to English.

?- spyg <Pred>. Spy predicate in DCG grammar (similar to spy).

?- spyr <Rule>. Spy Pragma translation rule

=== Parameters: ===

trace := N Tracelevel

N=0 just answers

N=1 generated TQL code (default)

N=2 FOL from parsing
Lexical analysis output

N=3 Lexical analysis output (more)
The parse tree
FOL from parsing (after anaphora resolution)

N=4 Print details of unknown words handling

maxdepth := N
Max depth of theorem prover
Default value N=3

spellcheck := 1. Set spell check on level 1 (one character errors)

busflag := true/false. Activate the Bus application interface.

tramflag := true/false. The trams are included

queryflag := true/false. Statements are implicit queries.

unknownflag := true/false. Accept/Reject unknown words

spellstreetflag := true/false. Accept spell correction in street names

textflag := true/false. Read text, process 1 sentence at a time
regardless of line divisions

smsflag := true/false. Short answers (<160 chars) are preferred.

smspermanentflag := true/false . Always smsflag

noparentflag := true/false. Skip parentheses including content

duallangflag := true/false. Try the other language if unknown words

noevalflag := true/false. Skip evaluation under TUC.

nodotflag := true/false. All '.' except last is ignored

dialog := 0 Missing data are replaced by default values
dialog := 1 Missing data left unknown (for prompting by dialog processor)

traceprog := 1 to 6 trace of pragmatic rule application

traceans := 1 to 6 trace of bus answer rule application

semantest := true/false separate error message if grammatical but meaningless.

parsetime_limit := 3000 (default) max parsing time in ms before parser gives up

language := english/norsk Default user language

unix_language := eng/nor Unix installation language

wold := <identifier> Possible to set another world than real

- NB World parameter is always reset to real after end of read file

=== System Commands in NL mode: ===

These commands starts with a backslash operator to distinguish them from NL text. The NL commands must appear on one line, a last dot is optional.

E: \ Exit, return to Prolog mode

E: \begin Same as reset, sets permanence on

E: \check Check the consistency of the base.

E: \clear Reset to initial condition.

E: \end Exit , turns permanence off

E: \c <file> Consults the file <file>.pl

E: \r <file> Reads text from the file <file>.e

E: \u <call> Shell command <call> is issued.

E: \<pred> Same as ?-call(<pred>).

E: \<pred> <arg> Same as ?-call(<pred>(<arg>)).

E: \<pred> <arg1> ... <argn>
Same as ?-call(<pred>(<arg1>,...,<argn>)).

All the System commands in Prolog mode listed above are also available in NL mode (e.g. the command \clear which is listed as an example).

== Language and Grammar ==

=== The accepted language ===

The accepted language, which is called E is a subset of English. A Norwegian version is also available.
Among all its restrictions, note the following:

- Punctuation is only allowed (taken into account) as the final marks, (no comma ,semicolon or hyphens).

statement .
question ?
command !

- A line may be split into several lines until the line ends with one of these marks.

- No attention is payed to the morphological form, eg. singular or plural or tense.

- All names not put in 'quotes' must be known by the system. New names can be defined using the standard phrase 'is a' <noun> or 'is an' <noun> as in
E: Anne is a girl.

- Genitive is written by a single 's' without apostrophe as in

E: who is johns mother ?

For a demonstration, see the content of the file 'twm.e' which is listed below.

\begin

Every person that gets a spot has this spot.

Once upon a time there lived in England a king.
The king had 3 wise men.
Every wise man got a colored spot on his forehead.
Every spot was red or white.
Every wise man could see every spot that was unequal
his own spot.
The king said that there was at least one white spot.
The king gave each wise man a white spot.

How did one man know that he had a white spot ?

\end

For your information, some files with extension '.e' are included in the delivery as illustrative examples. Also the file 'problems.e' contains examples of sentences which are not treated adequately.

=== The Grammar System (ConSensiCal Grammar) ===

The grammar is based on a simple grammar for statements, while questions and commands are derived by use of movements.
The grammar formalism is called ConSensiCal Grammar, which is an acronym for Context Sensitive Categorial Attribute Logical Grammar.
It is an easy to use variant of Extra-position Grammars, XG-grammars (F. Pereira), which is an extension of Definite Clause Grammars.
A characteristic grammatical expression in Consensical Grammar is found in the definition of a relative_clause w hich after 'that' expects a statement MINUS a noun_phrase.

A skeleton grammar follows below for declarative sentences (statements). The grammar which is listed in the file 'gram_e.pl' is much more comprehensive and sophisticated. The grammar is in fact an attributed
grammar that produces a formula in a first order event calculus.

=== Skeleton Grammar ===

sentence ---> statement . | question ? | command !

statement ---> noun_phrase verb_phrase

command ---> statement \ [you] .

verb_phrase ---> aux vp verb_complement(s)

aux ---> do | will | ...

vp ---> intransitive_verb | transitive_verb noun_phrase

verb_complement ---> prep_phrase | adverbial_phrase

prep_phrase ---> preposition noun_phrase

adverbial_phrase ---> today | yesterday | ...

noun_phrase ---> determiner adjective(s)
noun noun_complement(s)

noun_complement ---> prep_phrase | relative_clause

relative_clause ---> that (statement / noun_phrase)

determiner ---> a | the | every | ...

=== Meta-grammar rules ===

---> Production

| Alternatives

+ One or more occurrences.

* Zero or more occurrences.

... an open ended list .

C/D A phrase of category C, lacking a phrase of category C

C\D similar to / , but the subtracted phrase must occur first in the text.

C=D A phrase of category C is defined as if it was a D

C-D similar to / but D may reappear after C

[ ] Literal brackets

( ) just grouping parentheses

== Adaptability ==

TUC is adaptable, which means that there is a general grammar for the syntax, while the semantics of the words are declared in tables.

=== Dictionary ===

The dictionary is defined by the two files

dict_e.pl - contains the words definitions,

morph_e.pl - contains the morphological rules for English

Also, semantic.pl - contains many root forms of words.

In addition, lex.pl - contains general rules for lexical analysis.

The allowed words in the word classes

==== Nouns ====

Nouns are defined in a 'ako' hierarchy.
The hierarchy is tree-structured.

Definitions are made as facts of the kind

<class> ako <super-class>.

Example:

agent ako thing.
animate ako agent.
person ako animate.
animal ako animate.
adult ako person.
man ako adult.
father ako man.

==== Intransitive verb ====

Intransitive verbs are listed in the table

iv_templ(<verb>, <agent>).

Example:

iv_templ(live, animate).
iv_templ(work, employee).

==== Transitive verb ====

tv_templ(<verb>, <agent>, <patient>).

Example:

tv_templ(kill, animate, animate).
tv_templ(earn, person, money).

==== Adjectives ====

adj_templ(<adjective>,<class>).

Example:

adj_templ(dead,animate).
adj_templ(married,person).

==== Verb complements ====

Verb complements are modifiers of the verb.

v_compl(<verb>,<subject>,<preposition>,<object>).

Example:

v_compl(borrow,person,from,person).
v_compl(live,animate,in,time).

==== Noun complements ====

Noun complements are modifiers of the noun.

n_compl(<subject>,<preposition>,<object>)

Example:

n_compl(person,with,telescope).
n_compl(park,with,statue).

==== Adjective complements ====

Adjective complements are complements to the adjectives.

a_compl(<adjective>,<subject>,<preposition>,<object>)

Example:

a_compl(responsible,agent,for,thing).

==== Part-Of hierarchy ====

A relation apo (a part of) is used to define the constituent structure of the nouns.

Example:

month apo year.
week apo month.
day apo week.

This is used to lexically allow some expressions like

"week in a year"

without explicitly defining it as noun compliance.

NB. The relation apo must NOT be confused with the ako relation.

==== Attributes ====

The attributes of a class is declared by a
predicate 'has_a'.

<subject> has_a <object>

Example:

country has_a capital.
dog has_a owner.

==== Comparisons ====

Comparisons are polymorphic relations between objects.

comp_templ(<generic>,<class>,<class>,<specific>).

Example:

comp_templ(gt,person,person,height/gt).

=== User Defined Facts ===

A file 'facts.pl' contains a set of definitions of
object names. The main relation is 'is a' :

<name> isa <class>

richard isa employee.
january isa month.

=== Script files ===

TUC may be directed to read NL text from file.

The command is in dialogue mode

E: \r <file> .

The file <file> must have extension .e (English) or .n (Norwegian),

e.g. twm.e whose content is shown above.

=== Version Management Policy ===

The program system has a Version X.Y and a Date YYMMDD.
The Date is the date of the last modification.

The documentation also has a Version X.Y and a
Date YYMMDD, which is the date of the last correction.

The Version of the documentation and the program system
must correspond. The Date however may deviate.

Two files, documents or programs with the same
Version and Date are identical.

The documentation can be changed (improved) without
changing the Version number. If the program is unaffected,
only the Date is changed.

Similarly, programs can be changed (improved) with only
a change in the Date when the documentation is not affected

Each file has a similar field for the last revision date.
Changes in the files will normally be added a signature
and a date of the modification (e.g. TA-960702).

The date of the system release is the date of the
last revision of any of its files.
This date is written in the file version.pl.

== COPYRIGHT NOTICE ==

Tore Amble
Knowledge Systems Group
LINGIT A/S, Trondheim.
and the Department of Computer and Information Science

University of Trondheim (NTNU)
N-7491 Norway

Rune Sætre
Information Systems Group
Department of Computer and Information Science
University of Trondheim (NTNU)
N-7491 Norway

E-MAIL:
busstuc@idi.ntnu.no

== SOFTWARE DISCLAIMER ==

%% THIS IS THE END %%