ConTroll Reference Card

(for ConTroll v.1.0β and XTroll v.5.0β)

1 Basic Commands

To **start** ConTroll using the Xtroll interface, go to the directory containing the grammar and type `xtroll &` at the shell prompt. Two windows should open, the *Xtroll window* and the *Prolog window*.

To **exit** ConTroll, press button `Quit` in the Xtroll window.

To use the text version of ConTroll, start by typing `controll` instead of `xtroll` and exit using the ordinary `halt.` of prolog.

1.1 Compiling a grammar

A grammar minimally consists of two files, which by default are called `signature` and `theory`.

command	**effect**
compile_user_grammar	compiles the grammar

1.2 Interaction with the Signature

A signature defines the ontology, i.e. the types, their appropriate attributes, and the hierarchical order of types.

command	**effect**
show_approp(*<Type>*)	displays the appropriateness conditions of type *<Type>*
show_subtypes(*<Type>*)	displays the direct subtypes of type *<Type>*
show_all_subtypes(*<Type>*)	displays all subtypes of type *<Type>*
show_supertype(*<Type>*)	displays the direct supertype of type *<Type>*
show_all_subtypes(*<Type>*)	displays all supertypes of type *<Type>*
check(*<Desc>*)	tests description *<Desc>* for consistency with the signature

1.3 Interaction with the Grammar

We call a description **grammatical** iff it is consistent with the signature and described by each statement in the theory.

command	**effect**
show_constraint(*<Type>*)	displays the constraints defined on type *<Type>*
query(*<Desc>*)	tests description *<Desc>* for grammaticality prompts for more solutions
query_all(*<Desc>*)	like `query`, but directly supplies all solutions
inspect(*<Desc>*)	steps through the test for grammaticality of description *<Desc>*

2 Key bindings

Commands which prompt for user interaction before they are executed are marked with ....

2.1 XTroll window

delete selected data ...	`C-d`
de-iconify prolog window	`C-p`
quit XTroll ...	`C-q`

2.2 Prolog window

2.2.1 Motion

move forward over character	`C-f` or $\rightarrow$
move backward over character	`C-b` or $\leftarrow$
go to beginning of line	`C-a` or `Pos1` or `Home`
go to end of line	`C-e` or `End`

2.2.2 Command history

go backwards in command history	`C-p` or $\uparrow$
go forwards in command history	`C-n` or $\downarrow$

2.2.3 Text deletion

kill line after cursor	`C-k`
yank last kill	`C-y`
delete next character	`C-d` or Del
delete previous character	`C-h` or Backspace

If partial input has been provided in the command line, typing `C-p` will go backwards in those commands in the history which complete the input.

2.2.4 Miscellaneous

iconify prolog window	`C-z`
send interrupt to prolog	`C-c`

2.3 AVM/Tree windows

redraw	`C-l`
start text search from top...	`C-s`
start text search from bottom...	`C-r`
show next match	`C-n`
show previous match	`C-p`
hide attributes ...	`C-h`
expand reentrancies ...	`C-e`
quit this window ...	`C-q`

2.4 Debugger window

redraw	`C-l`
start text search from top...	`C-s`
start text search from bottom...	`C-r`
show next match	`C-n`
show previous match	`C-p`
hide attributes ...	`C-h`
expand reentrancies ...	`C-e`

3 Mouse Interaction

Apart from the selection of menu and submenu items, the following things can be done using the mouse.

3.1 XTroll window

- open an avm or tree window (the preferred representation set in Menu Options):
  double click with left mouse button on line in scroll window
- open an avm or tree window (the dispreferred representation):
  double click with right mouse button on line in scroll window
- delete a data package:
  select a line in scroll window using left mouse button, then type `C-d` or chose menu item *Delete selected data* from *Functions* menu.

3.2 Prolog window

- selecting text in the window:
  dragging mouse over text while pressing the left mouse button
- extending the selection:
  clicking in text to be included with left mouse button while holding the shift key
- pasting a selection into the prolog or some other window:
  middle mouse button
- note: clicking in the command line to position the cursor (currently) is not supported

3.3 AVM/Tree window

3.3.1 Middle mouse button

Holding the middle mouse button while moving the mouse moves the displayed picture in the window.

3.3.2 Left mouse button

Clicking the left mouse button over the different displayed elements pops up a menu displaying the possible interaction with this element:

- type:
  - Show hidden features (by selecting the corresponding subitem)
- attribute:
  - Hide/Show substructure
  - Select node as root
- reentrancy (numbered box):
  - Hide/Show substructure
  - Select node as root
- tree node:
  - toggle structure, i.e. show AVM below node
  - show structure (this window)
  - show structure (new window)

3.3.3 Right mouse button

Clicking the right mouse button has the following effect on the different elements displayed:

- attribute: hide attribute
- reentrancy (numbered box): open/close reentrancy
- tree node: show AVM corresponding to this node

3.4 Debugger window

3.4.1 Middle mouse button

Holding the middle mouse button while moving the mouse moves the displayed picture in the window.

3.4.2 Left mouse button

Clicking the left mouse button over the different displayed elements pops up a menu displaying the possible interaction with this element:

- type:
  - Show hidden features (by selecting the corresponding subitem)
- attribute:
  - Hide/Show substructure
  - Select node as root
- reentrancy (numbered box): executes command (unfold or creep) for the first matching clause of the selected goal in the debugger. Note that only clicking on goals corresponding to goals which can be executed will result in legal debugger commands. Others will only result in a warning in the Prolog window.
- tree node:
  - toggle structure, i.e. show AVM below node
  - show structure (this window)
  - show structure (new window)
- numbers displayed under **All**, **Fail**, **Delay**, **Det**:
  check which defining clauses match this goal, this information is shown in the clause display (horizontal window in the bottom right corner)
- the buttons **abort**, **fail**, **retry**, **det**, **prev**, **next**, **back**:
  select the corresponding debugger command
- the radio buttons **creep** and **unfold**:
  select the debugger action performed when clicking on a reentrancy
- numbers in clause display (horizontal window in the bottom right corner):
  executes command (unfold or creep) for the selected clause of the displayed goal in the debugger.

3.4.3 Right mouse button

Clicking the right mouse button has the following effect on the different elements displayed:

- attribute: hide attribute
- reentrancy (numbered box): open/close reentrancy
- tree node: show AVM corresponding to this node
- numbers under **ALL**,**Fail**, **Delay**, **Det**:
  show all matching clauses of selected goal
- numbers in clause display (horizontal window in the bottom right corner):
  show selected matching clause of the displayed goal in the debugger.

4 Syntax

The complete ConTroll syntax EBNF is listed at the end of this reference card.

4.1 Grammar

4.1.1 Principles

General well-formedness conditions on objects are expressed via implicational constraints. These generally take the form

```
<Desc1> ==> <Desc2>
```

where *<Desc2>* can be any description, and *<Desc1>* underlies the usual restrictions of descriptions in the scope of negation (no calls to relations and no variables). The meaning of such expressions is that *all* objects must satisfy them.

4.1.2 Relations

A specification of a definite relation comes in two parts: a compulsory type declaration specifying the types of the arguments, and a set of clauses.

```
<Relname>(<Arg1-Type>,...,<ArgN-Type>) **> <Arg0-Type>.
<Relname>(<Arg1>,...,<ArgN>) := <Arg0>.
```

4.2 Descriptions

Feature descriptions are built out of four basic categories:

1. Variables – all Prolog variables are also ConTroll variables. As a rule of thumb, this is anything that starts with a capital letter. Variables play the role of HPSG tags.
2. Types – these are the things you have declared in your type hierarchy.
3. Attributes – also called features. Everything followed by a colon is a feature. All features must also be declared in the type hierarchy.
4. Relations – express relational dependencies between objects.

4.2.1 Basic

`t`	object has type *t*
`f:t`	value of attribute F is type `t`
`f:g:h:t`	value of path F|G|H is type `t`
`f:X,g:X`	structure sharing values of attributes F and G

4.2.2 Complex

If *<D>* and *<E>* are descriptions, so are the following.

`~ <D>`	negation
`<D> , <E>`	conjunction
`<D> ; <E>`	disjunction

4.3 Delays

A delay statement postpones the execution of the call to *<Goalname>* until the call is at least as specific as *<DelayDesc>*:

```
delay(<Goalname>, <DelayDesc>).
```

- *<DelayDesc>* is a special kind of description, where
  - the special list syntax ([], ...) is disallowed,
  - the additional unary operator @ can be used, and
  - the additional binary operator * can be used.
- If a type in *<DelayDesc>* is prefixed by the new operator @, the corresponding type in the call has to be more specific than the one provided.
- The binary operator * is used to specify a description under a path of unknown length:

  ```
  (<Path>)*<Desc>
  ```

  - <Path> is a list of features separated by :
  - <Desc> a delay description without *
- If several delay statements exist for one goal, all of them have to be satisfied before a call to that goal can be executed.
- If during a calculation only delayed goals remain, then ConTroll fails this computation and reports
  no more undelayed goals. By choosing one of the options of the menu item "remaining delays action" in the "options" menu, it is possible to start the debugger at this point.

4.4 Special lexicon syntax

In principle, the lexicon in ConTroll has no special status. A typical HPSG lexicon is an implicative constraint on objects of type *word*. The consequent of this constraint is a disjunction of descriptions, where each description is a *lexical entry*. Such an HPSG lexicon can be directly expressed in a ConTroll grammar in the following way, where the *<LE>* represent the different lexical entries:

```
word ==> <LE1> ; <LE2> ; ...  ; <LEn> .
```

Since encoding the lexicon in such a disjunction is notationally impractical (and most Prologs only allow terms of limited length), ConTroll offers a special notation to specify lexical entries. This notation consist of three specifications:

- Each lexical entry is specified as a statement of the form:

  <Desc1> ---> <Desc2>.

- How the two descriptions to the left and right of the `--->` operator are mapped into an ordinary description, i.e., a description like the <*LE*> in the traditional HPSG lexicon we saw above, is determined by including one statement of the following form in a theory file:

  ```
  lexicon_style(<Var1>,<Var2>,<Desc>).
  ```

  <*Var1*> is a variable corresponding to the left-hand side of each `--->` statement, and <*Var2*> to the right-hand one. Using these variables, the description <*Desc*> specifies how the left-hand side and the right-hand side are supposed to be combined to form an ordinary lexical entry.

- A statement to declare for which type of object the new notation is intended to introduce the implicational constraint encoding the lexicon. For example, in a traditional HPSG lexicon, objects of type *word*. One such declaration of the lexicon type obeying the following syntax should be included in a theory file:

  ```
  lexicon_type(<Type>).
  ```

This special notation is compiled exactly like an ordinary implicative statement on the type defined to be the lexicon type. Note that this means that additional implicative constraints on the lexicon type are interpreted conjunctively in the usual way!

4.5 EBNF syntax specification

```
Program          ::=  ε | ProgramStat Program
ProgramStat      ::=  Implication. | DefSpec | ControlStat.
Implication      ::=  FeatureTerm ==> FeatureTerm |
                      ⟨ImplicationID⟩ === FeatureTerm ==> FeatureTerm
DefSpec          ::=  TypeDecl Clauses
TypeDecl         ::=  PredSpec **> ⟨Type⟩.
PredSpec         ::=  ⟨PredName⟩ | ⟨PredName⟩( TypeVector )
TypeVector       ::=  ⟨Type⟩ | ⟨Type⟩, TypeVector
Clauses          ::=  ε | Clause Clauses
Clause           ::=  Atom := FeatureTerm.
Atom             ::=  ⟨PredName⟩ | ⟨PredName⟩( TermVector )
TermVector       ::=  FeatureTerm | FeatureTerm, TermVector
FeatureTerm      ::=  ⟨Type⟩ | ⟨Variable⟩ | Atom |
                      ⟨Feature⟩: FeatureTerm | ~ FeatureTerm |
                      FeatureTerm , FeatureTerm |
                      FeatureTerm ; FeatureTerm |
                      (FeatureTerm) | [] | [ TermVector ]|
                       [ TermVector | FeatureTerm ]
ControlStat      ::=  Delay | TermExp | FeatureExp | TreeSpec | AccessRelSpec
Delay            ::=  delay( ⟨Type⟩ , DelayTerm )|
                      delay( ⟨PredName⟩ , DelayTerm )|
                      delay_deterministic( ⟨Type⟩ )|
                      delay_deterministic( ⟨PredName⟩ )
DelayTerm        ::=  BasicDelayTerm | StarDelayTerm | (DelayTerm)
BasicDelayTerm   ::=  ⟨Type⟩ | @ ⟨Type⟩ | ⟨Feature⟩: DelayTerm
                      DelayTerm , DelayTerm | DelayTerm ; DelayTerm
StarDelayTerm    ::=  Path * BasicDelayTerm | Path :( Path * BasicDelayTerm )
TermExp          ::=  ⟨Type⟩ << ⟨Type⟩.
FeatureExp       ::=  ⟨Feature⟩ <<< ⟨Feature⟩.
TreeSpec         ::=  ptree_desc( ⟨PrologAtom⟩ , FeatureTerm ). |
                      ptree_cat( FeatureTerm ). | ptree_phon( ⟨Feature⟩ ).
AccessRelSpec    ::=  access_suffix( ⟨PrologAtom⟩ , ⟨Type⟩ ). |
                      access_rule( ⟨Type⟩ , Path , ⟨Type⟩ ).
Path             ::=  ⟨Feature⟩ | ⟨Feature⟩ : Path
```