HETS User Guide
– Version 0.3 –

Till Mossakowski

Department of Computer Science
and Bremen Institute for Safe Systems,
University of Bremen, Germany.

Comments to: hets@tzi.de

February 17, 2004

1 Introduction

The Heterogeneous Tool Set (HETS) is the main analysis tool for the specification language heterogeneous CASL. Heterogeneous CASL (HETCASL) combines the specification language CASL with CASL extensions and sublanguages, as well as completely different logics and even programming languages such as Haskell. HETCASL extends the structuring mechanisms of CASL: *Basic specifications* are unstructured specifications or modules written in a specific logic. The graph of currently supported logics is shown in Fig. 1, and the degree of support by HETS in Fig. 2.

With *heterogeneous structured specifications*, it is possible to combine and rename specifications, hide parts thereof, and also translate them to other logics. *Architectural specifications* prescribe the structure of implementations. *Specification libraries* are collections of named structured and architectural specifications.

HETS consists of logic-specific tools for the parsing and static analysis of the different involved logics, as well as a logic-independent parsing and static analysis tool for structured and architectural specifications and libraries. The latter of course needs to call the logic-specific tools whenever a basic specification is encountered.

An introduction to CASL can be found in the CASL User Manual [3]; the detailed language reference is given in the CASL Reference Manual [9]. These documents explain both the CASL logic and language of basic specifications as well as the logic-independent constructs for structured and architectural specifications. Corresponding documents explaining the HETCASL language constructs for *heterogeneous* structured specifications are forthcoming, until then, [7] may

$$\begin{tikzcd}
\text{Haskell} & \text{HasCasl} & \text{Csp-Casl} \\
& \text{Casl} \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 1: Graph of logics currently supported by Hets.

Language	Parser	Static Analysis	Prover
Casl	x	x	-
HasCasl	x	(x)	-
Haskell	x	x	-
Csp-Casl	(x)	-	-
Structured specifications	x	x	(x)
Architectural specifications	x	-	-

Figure 2: Current degree of Hets support for the different languages.

serve as a short introduction. Moreover, the main heterogeneous constructs will be illustrated in Sect. 4 below.

An overview of HasCasl is given in [13]; the language is summarized in [12]. The latter document is also available on the CD-ROM in `Tools/Hets/doc`. The definitive reference for Haskell is [10], see also `www.haskell.org`. An introduction to Csp-Casl is given in [11].

2 Getting started

The latest Hets version can be obtained from the CoFI tools home page

```
http://www.cofi.info/Tools
```

Since Hets is being improved constantly, it is recommended always to use the latest version.

Hets currently is available for Linux, Solaris and Mac OS-X. You can either start the script `Tools/Hets/hets` (the script automatically finds the binary for your architecture). Or you can install Hets on your local file system (which is strongly recommended, since only then Hets can write intermediate `.env` files that are needed for compilation of library downloads). This is done by opening a shell, `cd`ing into `Tools/Hets`on the CD-ROM[1] and then calling

```
./install <dir>
```

where `<dir>` is the directory where Hets should be installed.

MacIntosh users need to install some libraries from `Tools/Hets/hets/macintosh`

[1]For Linux and Solaris users: you may need to mount the CD-ROM first. You can lookup in `/etc/fstab` where it is mounted, typcial locations are `/media/cdrom` or `/mount/cdrom`. For MacIntosh users: cd into `/Volumes` and look there for the CD-ROM.

```
    cp libreadline.4.3.dylib /usr/local/lib
    tar xzf Haskell-libs.tgz -C /Library/Frameworks
```

The `install` script installs these automatically for you. Note that you may have to be `root` in order to be able to do this. See

`http://www.osxfaq.com/Tutorials/Root_User_Creation/index.ws`

for instructions how to create a root user.

If you want to compile HETS from the sources, please follow the instructions of the `Tools/Hets/src/INSTALL` file.

daVinci

For the display of development graphs, HETS uses daVinci. daVinci is maintained by b-novative and b-novative holds all rights. This release may be accompanied with free binary releases of daVinci Version 2.1, but daVinci 2.1 is no longer supported or maintained. However, you are encouraged to obtain the latest version of daVinci Presenter (currently 3.0.5) from `http://www.b-novative.com`. It needs a license key, but is free for academic purposes.

Linux: the Linux binary release of daVinci 2.1 relies on the old shlibs5 that must be installed on your system (if ./daVinci cannot be found/executed, then shlibs5 are missing)

Solaris: the Solaris binary release of daVinci 2.1 should work without problems

Macintosh (Darwin): there is no release of daVinci for Macintosh, but b-novative may have one in the mean time

For best quality, get the latest version of daVinci from b-novative.

Environment variables

The script in `Tools/Hets/hets` sets the following environment variables:

For HETS to find daVinci, the environment variables `DAVINCIHOME` and `UNIDAVINCI` must be set to the installation directory of daVinci and to the actual executable, respectively.

```
export DAVINCIHOME=<path>/daVinci_V2.1
export UNIDAVINCI=<path>/daVinci_V2.1/daVinci
```

The environment variable `HETS_LIB` should be set to a location where specification libraries are stored. By default, it is set to the `Libraries` directory on the CD-ROM (or the corresponding copy on your local file system).

3 Analysis of Specifications

Consider the following CASL specification:

spec STRICT_PARTIAL_ORDER =

sort *Elem*
pred $\_\_ < \_\_ : Elem \times Elem$
$\forall x, y, z : Elem$
- $\neg(x < x)$ %(strict)%
- $x < y \Rightarrow \neg(y < x)$ %(asymmetric)%
- $x < y \wedge y < z \Rightarrow x < z$ %(transitive)%

%{ Note that there may exist x, y such that
neither $x < y$ nor $y < x$. **}%**

end

Hets can be used for parsing and checking static well-formedness of specifications.

Let us assume that the example is in a file named `Order.casl` (actually, this file is provided with the Hets distribution). Then you can check the well-formedness of the specification by typing (into some shell):

```
hets Order.casl
```

Hets checks both the correctness of this specification with respect to the Casl syntax, as well as its correctness with respect to the static semantics (e.g. whether all identifiers have been declared before they are used, whether operators are applied to arguments of the correct sorts, whether the use of overloaded symbols is unambiguous, and so on). The following flags are available in this context:

`-p`, `--just-parse` Just do the parsing – the static analysis is skipped.

`-s`, `--just-structured` Do the parsing and the static analysis of (heterogeneous) structured specifications, but leave out the analysis of basic specifications. This can be used to quickly produce a development graph showing the dependencies among the specifications (cf. Sect. 5).

`-L DIR`, `--hets-libdir=DIR` Use `DIR` as the directory for specification libraries (equivalently, you can set the variable `HETS_LIB` before calling Hets).

4 Heterogeneous Specification

Hets accepts plain text input files with the following endings:

Ending	default logic	structuring language
`.casl`	Casl	Casl
`.het`	Casl	Casl
`.hs`	Haskell	Haskell

Although the endings `.casl` and `.het` are interchangeable, the former should be used for libraries of homogeneous Casl specifications and the latter for HetCasl libraries of heterogeneous specifications (that use the Casl structuring constructs). Within a HetCasl library, the current logic can be changed e.g. to HasCasl in the following way:

```
logic HasCASL
```

The subsequent specifications are then parsed and analysed as HasCasl specifications. Within such specifications, it is possible to use references to named Casl specifications; these are then automatically translated along the default embedding of Casl into HasCasl (cf. Fig. 1). (Heterogeneous constructs for explicit translations between logics will be made available in the future.)

The ending `.hs` is available for directly reading in Haskell programs, and hence supports the Haskell module system. By contrast, in HetCasl libraries (ending with `.het`), the logic Haskell has to be chosen explicitly, and the Casl structuring syntax needs to be used:

```
library Factorial

logic Haskell

spec Factorial =
fac :: Int -> Int
fac n = foldl (*) 1 [1..n]
end
```

Note that according to the Haskell syntax, Haskell function declarations and definitions need to start with the first column of the text.

5 Development Graphs

Development graphs are a simple kernel formalism for (heterogeneous) structured theorem proving and proof management. A development graph consists of a set of nodes (corresponding to whole structured specifications or parts thereof), and a set of arrows called definition links, indicating the dependency of each involved structured specification on its subparts. Arising proof obligations are attached as so-called theorem links to this graph. Details can be found in the Casl Reference Manual [9, IV:4].

The following options let Hets show the development graph of a specification library:

`-g, --gui` Shows the development graph in a GUI window.

`-G, --only-gui` Shows the development graph in a GUI window, and suppresses the writing of an output file.

Here is a summary of the types of nodes and links occurring in development graphs:

Named nodes correspond to a named specification.

Unnamed nodes correspond to an anonymous specification.

Elliptic nodes correspond to a specification in the current library.

Rectangular nodes are external nodes corresponding to a specification downloaded from another library.

Black links correspond to reference to other specifications (definition links in the sense of [9, IV:4]).

Blue links correspond to hiding (hiding definition links).

Red links correspond to open proof obligations (theorem links).

Green links correspond to proved proof obligations (theorem links).

Solid links correspond to global (definition or theorem) links in the sense of [9, IV:4].

Dashed links correspond to local (definition or theorem) links in the sense of [9, IV:4].

We now explain the menus of the development graph window. Most of the pull-down menus of the window are daVinci-specific layout menus; their function can be looked up in the daVinci documentation. The exception is the Edit menu. Moreover, the nodes and links of the graph have attached pop-up menus, which appear when clicking with the right mouse button.

Edit This menu has two submenus:

- **Unnamed nodes** With the “Hide” submenu, it is possible to reduce the complexity of the graph by hiding all unnamed nodes; only nodes corresponding to named specifications remain displayed. Paths between named nodes going through unnamed nodes are displayed as links. With the “Show” submenu, the unnamed nodes re-appear.
- **Proofs** This menu allows to apply some of the deduction rules for development graphs, see Sect. IV:4.4 of the CASL Reference Manual [9]. However, it is still experimental and far from being completely implemented. Moreover, in the future HETS will be interfaced with various logic-specific theorem proving, rewriting and consistency checking tools.

Pop-up menu for nodes Here, the number of submenus depends on the type of the node:

- **Show signature** Shows the signature of the node.
- **Show sublogic** Shows the logic and, within that logic, the minimal sublogic for the signature and the axioms of the node.
- **Show origin** Shows the kind of CASL structuring construct that led to the node.

Show just subtree (Only for named nodes) Reduce the complexity of the graph by just showing the subtree below the current node.

Undo show just subtree (Only for named nodes) Undo the reduction.

Show referenced library (Only for external nodes) Open a new window showing the development graph for the library the external node refers to.

Pop-up menu for links Again, the number of submenus depends on the type of the node:

Show morphism Shows the signature morphism of the link. It consists of two components: a logic translation and a signature morphism in the target logic of the logic translation. In the (most frequent) case of an intra-logic signature morphism, the logic translation component is just the identity.

Show origin Shows the kind of CASL structuring construct that led to the link.

Show proof status (Only for theorem links) Show the proof status.

6 Reading, Writing and Formatting

HETS provides several options controlling the types of files that are read and written.

`-i ITYPE`, `--input-type=ITYPE` Specify `ITYPE` as the type of the input file. The default is `het` (HETCASL plain text). `ast` is for reading in abstract syntax trees in ATerm format, while `ast.baf` reads in the compressed ATerm format.

`-O DIR`, `--output-dir=DIR` Specify `DIR` as destination directory for output files.

`-o OTYPES`, `--output-types=OTYPES` `OTYPES` is a comma separated list of output types:

```
 (ast|[fh]?dg(.nax)?).(het|trm|taf|html|xml)
|(pp.(het|tex|html))
|(graph.(dot|ps|davinci))
```

The default is `dg.taf`, which means that the development graph of the library is stored in textual ATerm format (`taf`). This format can be read in when a library is downloaded from another library, avoiding the need to re-analyse the downloaded library.

The `pp` format is for pretty printing, either as plain text (`het`), LaTeXinput (`tex`) or HTML (`html`). A formatter with pretty-printed output currently is available only for the CASL logic. For example, it is possible to generate a pretty printed LaTeX version of `Order.casl` by typing:

```
hets -o pp.tex Order.casl
```

This will generate a file `Order.pp.tex`. It can be included into LaTeX documents, provided that the style `hetcasl.sty` coming with the Hets distribution (`LaTeX/hetcasl.sty`)) is used.

The option `-r RAW` or `--raw=RAW` allows one to influence the formatting in more detail. Here, `RAW` is `(ascii|text|(la)?tex)=STRING`, and `STRING` is passed to the appropriate pretty-printer. The deatils of these options are to be fixed in the future only.

The other output formats are for future usage.

7 Miscellaneous Options

`-v[Int]`, `--verbose[=Int]` Set the verbosity level according to `Int`. Default is 1.

`-q`, `--quiet` Be quiet – no diagnostic output at all. Overrides -v.

`-V`, `--version` Print version number and exit.

`-h`, `--help`, `--usage` Print usage information and exit.

`+RTS IntM -RTS` Increase the stack size to `Int` megabytes (needed in case of a stack overflow). This must be the first option.

8 Limits of Hets

Hets is still intensively under development. In particular, the following points are still missing:

- There is no static analysis for architectural specifications.
- Distributed libraries are always downloaded from the local disk, not from the Internet.
- Version numbers of libraries are not considered properly.
- The proof engine for development graphs has not been completed.

Hets has been built based on experiences with its precursors, Cats and Maya. The Casl Tool Set (Cats) [6, 8] comes with similar analysis tools as Hets (only for Casl). Cats should be used if a static analysis of architectural specifications is needed, because this is not available in Hets yet. The management of development graphs is not integrated in Cats, but is provided with a stand-alone version of the tool Maya [2, 1]. Cats and Maya can be obtained from the CoFI tools home page [5].

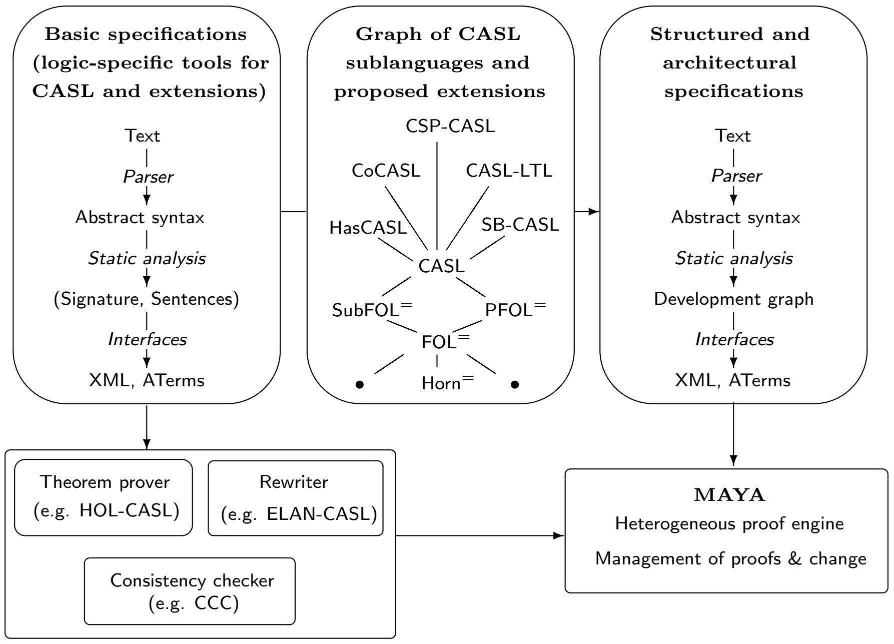
Basic specifications (logic-specific tools for CASL and extensions)
Text
Parser
Abstract syntax
Static analysis
(Signature, Sentences)
Interfaces
XML, ATerms
Graph of CASL sublanguages and proposed extensions
CSP-CASL
CoCASL
CASL-LTL
HasCASL
SB-CASL
CASL
SubFOL=
PFOL=
FOL=
Horn=
Structured and architectural specifications
Text
Parser
Abstract syntax
Static analysis
Development graph
Interfaces
XML, ATerms
Theorem prover (e.g. HOL-CASL)
Rewriter (e.g. ELAN-CASL)
Consistency checker (e.g. CCC)
MAYA
Heterogeneous proof engine
Management of proofs & change

Figure 3: Architecture of the heterogeneous tool set.

9 Architecture of Hets

Hets is written in Haskell. Its parser uses combinator parsing. The user-defined (also known as “mixfix”) syntax of Casl calls for a two-pass approach. In the first pass, the skeleton of a Casl abstract syntax tree is derived, in order to extract user-defined syntax rules. In a second pass, which is performed during static analysis, these syntax rules are used to parse any expressions that use mixfix notation. The output is stored in the so-called ATerm format [4], which is used as interchange format for interfacing with other tools.

Hets provides an abstract interface for institutions, so that new logics can be integrated smoothly. In order to do so, a parser, a static checker and a prover for basic specifications in the logic have to be provided.

The architecture of Hets is shown in Fig. 3.

If your favourite logic is missing in Hets, please tell us (hets@tzi.de). We will take account your feedback when deciding which logics and proof tools to integrate next into Hets. Help with integration of more logics and proof tools into Hets is also welcome.

Hets is mainly maintained by Christian Maeder (maeder@tzi.de) and Till Mossakowski (till@tzi.de). The mailing list is hets@tzi.de.

Acknowledgement Hets has been programmed by the following people:

Katja Abu-Dib, Carsten Fischer, Jorina Freya Gerken, Sonja Gröning, Wiebke Herding, Martin Kühl, Klaus Lüttich, Christian Maeder, Maciek Makowski, Till Mossakowski, Daniel Pratsch, Felix Reckers, Markus Roggenbach, and Pascal Schmidt.

HETS also benefited much from contributions by Serge Autexier, Dieter Hutter and Bartek Klin (through work on its precursors CATS and MAYA).

References

[1] S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The development graph manager MAYA (system description). In H. Kirchner and C. Ringeissen, editors, *Algebraic Methods and Software Technology, 9th International Conference, AMAST 2002, Saint-Gilles-les-Bains, Reunion Island, France, Proceedings*, LNCS Vol. 2422, pages 495–502. Springer, 2002.

[2] Serge Autexier and Till Mossakowski. Integrating HOL-CASL into the development graph manager MAYA. In A. Armando, editor, *Frontiers of Combining Systems, 4th International Workshop, FroCoS 2002, Santa Margherita Ligure, Italy, Proceedings*, LNCS Vol. 2309, pages 2–17. Springer, 2002.

[3] Michel Bidoit and Peter D. Mosses. *CASL User Manual*, volume 2900 of *Lecture Notes in Computer Science*. Springer, 2003. To appear.

[4] M. G. J. van den Brand, H. A. de Jong, P. Klint, and P. Olivier. Efficient annotated terms. *Software, Practice & Experience*, 30:259–291, 2000.

[5] CoFI (The Common Framework Initiative) Tools Group. Home page. `http://www.cofi.info/Tools`.

[6] Till Mossakowski. CASL: From semantics to tools. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Berlin, Germany, Proceedings*, LNCS Vol. 1785, pages 93–108. Springer, 2000.

[7] Till Mossakowski. Foundations of heterogeneous specification. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT 2002, Frauenchiemsee, Germany, 2002, Revised Selected Papers*, LNCS Vol. 2755, pages 359–375. Springer, 2003.

[8] Till Mossakowski, Kolyang, and Bernd Krieg-Brückner. Static semantic analysis and theorem proving for CASL. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, 1997, Selected Papers*, LNCS Vol. 1376, pages 333–348. Springer, 1998.

[9] Peter D. Mosses, editor. *Casl Reference Manual*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004. To appear.

[10] S. Peyton-Jones, editor. *Haskell 98 Language and Libraries — The Revised Report*. Cambridge, 2003. also: J. Funct. Programming **13** (2003).

[11] Markus Roggenbach. CSP-Casl – A new integration of process algebra and algebraic specification. In F. Spoto, G. Scollo, and A. Nijholt, editors, *Algebraic Methods in Language Processing, AMiLP 2003*, TWLT Vol. 21, pages 229–243. Univ. of Twente, 2003.

[12] L. Schröder, T. Mossakowski, and C. Maeder. HasCasl – Integrated functional specification and programming. Language summary. Available at `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/HasCASL`

[13] Lutz Schröder and Till Mossakowski. HasCasl: Towards integrated specification and development of Haskell programs. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methods and Software Technology, 9th International Conference, AMAST 2002, Saint-Gilles-les-Bains, Reunion Island, France, Proceedings*, LNCS Vol. 2422, pages 99–116. Springer, 2002.