An Evaluation of the State of the **CASA** System[1]

Ralf Hemmecke,
Erik Hillgarter,
Wolfgang Schreiner,
Franz Winkler

Research Institute for Symbolic Computation, Linz

October 27, 1998

[1]Supported by FWF, Project HySaX, P11160-TEC and SFB F013, Project 1304

Abstract

The computer algebra software **CASA** (**C**omputer **A**lgebra **S**oftware for **A**lgebraic Geometry), which is based on the computer algebra system **Maple**, is being developed by the computer algebra group at RISC under the direction of F. Winkler. In this report, **CASA** is analyzed with respect to its current state and possible improvements. Finally, some improvements and extensions of the system are outlined.

Contents

Chapter 1

Introduction to the Project and its Goals

In this introduction, we give some information about **CASA** and the environment and goals of this project.

1.1 What is CASA?

CASA is a special-purpose system for computational algebra and constructive algebraic geometry.

Constructive methods in algebra and algebraic geometry have been gaining more and more importance with the availability of computers and computer algebra software. Since its first version (see [1]), **CASA** has been designed to perform computations and utilize reasoning about algebraic and geometrical objects in the classical affine and projective spaces over algebraically closed fields of characteristic zero. More precisely, the field has to be a computable field in the sense of the underlying computer algebra system **Maple**, i.e.,all the arithmetic operations have to be available in the system. Usually, the field of computation is the rational numbers $\mathbf{Q}$ or a finite algebraic extension thereof.

The system has been developed since 1990. With the latest version (version 2.3) **CASA** is the ongoing product of the Computer Algebra Group at the Research Institute for Symbolic Computation (RISC-Linz), the University of Linz, Austria, under the direction of Franz Winkler. The system is built on the kernel of the widely used computer algebra system **Maple**. (**CASA** runs on **Maple** V.4. An older version of **CASA**—version 2.3 ALPHA—is still available for **Maple** V.3).

Several people have contributed to the system in one way or another. The names of the main contributors (see [12, 4], [11], [3], [5, 6, 8, 7] and [10]) are K. Aigner, R. Gebauer, E. Hillgarter, M. Kalkbrener, M. Mňuk, J. R. Sendra, P. Stadelmeyer, Q.-N. Tran, B. Wall, F. Winkler (director).

The major goal of **CASA** is to provide a comfortable, easy to use, efficient, flexible and mathematically exact working environment for computational algebra and constructive algebraic geometry, where all basic theoretical concepts map easily to available data structures. **CASA** is built on the kernel of **Maple** and is fully independent of the operating system; hence, it can be used on every hardware where Maple is running.

There are software systems which partially cover some of the fields of **CASA**, however, no other system seems to provide all the functionalities of **CASA**.

1.2 Environment and Goals of the Project

The software system **CASA** should provide easy installation, portability and expandability.

The goal of this report is to provide a solid basis for the future development of **CASA** by a thorough analysis of its current state (including browsing through the system, finding inconsistencies, redundancies, ...). For achieving this goal, it became evident that the followig problems have to be attacked:

- As the most important point, an improvement of the quite complex system structure seemed necessary. The system architecture had to be explored by browsing through the system (including such diverse topics as installation procedure, documentation facilities, **CASA** data types, ...) and through the **CASA** directories (including topics like package dependencies and the use of private directories). Problems arose from inconsistencies and redundancies in the **CASA** code.
- "Unisourcification" of documentation files: the current version of **CASA** contains three main documentation sources: the web page, the online documentation, and the **CASA** Reference Manual [9]. These three documentation forms are not generated from one source; hence they do not necessarily correspond exactly to each other. In addition, future changes and updates have to be carried out three times. In order to overcome these deficiencies, the generation of all three documentation forms from one source seemed to be desirable.
- Simplification of the installation procedure for **CASA**.
- Creating a test-suite for the automatic testing of new packages and **CASA** or **Maple** versions.
- Improving the efficiency of data representation of algebraic sets.
- Documenting how to write a **CASA** package.

In Chapter 2, the outcome of the analysis of the system is described. In Chapter 3, the description of future goals based on the results outlined in Chapter 2 is presented.

Chapter 2

Description of Current State

In this chapter, we explain the current state of **CASA**. This includes the description of the file/directory structure of **CASA**, conventions concerning the code, the use of types, and dependencies between different packages. There is also information on how to make and start up **CASA**.

2.1 Files (Directory Structure)

In this section, the structure of the installation directory of **CASA** is outlined. Additionally, the unused files are identified and listed.

2.1.1 The Directory Structure

This subsection outlines the structure of the **CASA** installation directory. In this directory, we find the following files and directories:

```
GPL            README  help.maple   maplebugs
INSTALL        demo    help.maple~  mgrobner
Makefile       doc     lib          src
Makefile.defs  help    libV3        www
```

A short description of some relevant files/directories follows:

Text Files

This sub-subsection discusses the text files occurring in the installation directory.

- `GPL` (text file): **G**nu General **P**ublic **L**icence.

- `INSTALL` (text file): Documentation on the installation of **CASA**. Since **CASA** is implemented as a **Maple** library, the document concentrates on the following issues:
  - which directories are to be created,
  - which files have to be copied to those directories,
  - which paths have to be set.
- `README` (text file): general documentation on **CASA**. It deals with the following topics:
  - introduction
    - 4 different representations of algebraic sets
    - classes of available **CASA** operations
  - What is new in **CASA** 2.3?
  - installation of **CASA** on different types of systems
  - contents of this distribution
  - further developments and updates
  - copyright notice

Demos, Documentation, Help

In this sub-subsection, we describe the directories that contain files for **Maple**-demos, help pages and printed documentation.

- `demo` (directory): Some **Maple** work sheets presenting **CASA** facilities, e.g. `casa-demo.mws` (contains text that also appears in `README`).
- `doc` (directory): Documentation of **CASA**. Contains many `.tex`-, `.ps-`, and `.sty`-files that are used by `manual.tex` (the LaTeX-file for the **CASA** reference manual Version 2.3 [9]). Other files of interest are:
  - Some perl programs: `cnv2pp, doman, mklm, mkman, mkmaplehelp`
  - Some Lisp programs: `mkhelp.el, refman.el`
  - `rmLaTeXsrc`: removes LaTeX (source-) files; should be generated!?
  - `tpp`: The TeX Preprocessor

  There exists the subdirectory `mgrobner`, containing the `mgrobner.tex` and several related `.tex`-files that constitute the documentation for Peter Stadelmeyers `mgrobner`-package ("Projektbericht zum Programmierpraktikum Computer Algebra 93"). Some `.txt`-files provide examples.
- `help` (directory): the **Maple** help pages for **CASA** (text and `.m`-files). The directory structure looks as follows (all leaves are directories):

```
help
    casa
        mws
```

```
        src
        text
    text
```

Some words on these directories:

- `help/text`: Contains the file `casa.m` that is responsible for some text that appears after starting **CASA**.
- `help/casa/mws`: This directory contains two files for (nearly) every **Maple** command:

  commandname,

  comandname`.mws`.

  The *comandname*`.mws`-files (**Maple** work sheets) are generated help files that are used for the **Maple** online help (i.e. help that appears after typing ?*commandname* within the **Maple** environment). The *commandname*-files contain **Maple** help files in some handwritten format, but those files are not the source for the generated *comandname*`.mws`-files, and hence are superfluous.

  The exceptions from the above given scheme are:

attribute	Attributes.mws
	RPHcurve.mws
	dixon.mws
	extNewton.mws
	implOffset.mws
mvresultant	
	pacplot.mws
	paraOffset.mws
passCurve	
	realroot_a.mws
	realroot_sb.mws
Template.txt	Template.mws

- `help/casa/src`: Source files for the `.mws` files. The exceptions are:

Gwalk GWalk.mws	Template.mws	casa casa.mws

- `help/casa/text`: empty!

• `help.maple` (file): Script for the automatic processing of help files.
The header contains the following information:

```
############################################################
# NOTE: run the following commands from an X-window session
############################################################
#--rhx:
#-- You need to adjust the variable CASATOP to the
#-- directory where you want this script to take and
#-- place the files. (see below)
```

```
#--
#-- This adjustment is very bad. Should find a better
#-- solution. So that this file need not be changed.
###########################################################
```

- `www` (directory): files for **CASA** web page.

  This directory contains many `.GIF`-, `.gif`- and `.gz`-files that constitute the text and the pictures for the **CASA** web page. The other files are among:

  `CASAman.sty, casa.log, casa.tex, casa02.jpg, casa03.jpg, comay02.jpg, index, index.cache, index.log, index.tex, mydef.sty, texput.log.`

System Files

This sub-subsection deals with the system files.

- `lib` (directory): the **Maple** library with. Directory structure:

  `lib`
  - `mapl.hdb` (generated file)
  - `maple.ind` (generated file)
  - `maple.lib` (generated file)
  - `update` (empty directory)

- `libV3` (directory): Former version of `lib`. Directory structure:

  `libV3`
  - `maple.ind` (generated file)
  - `maple.lib` (generated file)
  - `update` (empty directory)

- `Makefile.defs` (file): Contains definitions for `Makefile`.

Maple Code

In this sub-subsection we describe the directories that contain **Maple** code.

- `maplebugs`(directory): Directory structure (all leaves are files):

```
maplebugs
        Makefile
        Makefile.defs
        evala
             AFactors
```

```
            Makefile
            intersect
            intersect.m (generated file)
        Makefile
```

- `mgrobner` (directory): Package for computation in modules over polynomial rings. The corresponding help pages can be found in `mgrobner/help`. Directory structure:

```
mgrobner
        Makefile
        Makefile.defs
        help
            mgrobner
                    src
                       5 help files
                    text
                        5 help files.m
            text
                    mgrobner
                    mgrobner.m
        src
            Makefile
            mapleprogram
            mapleprogram.m
            ⋮
            [directory]
                    Makefile
                    mapleprogram
                    mapleprogram.m
                    ⋮
```

- `src` (directory): **Maple** code sources of **CASA**. Directory structure:

```
src
   Makefile
   directory
           Makefile
           mapleprogram
           mapleprogram.m
           ⋮
           [Private]
                   Makefile
                   mapleprogram
                   mapleprogram.m
                   ⋮
```

directory

⋮

`src` contains the following subdirectories that constitute **Maple** packages:

```
curves     extnewton  init       oldStuff   radical
setops     basics     decompose  geoprover  misc
plot       realroot   convert    dimension  grobner
offset     private    resultant  type
```

2.1.2 Unused Files

We list here all the files which are not used by the makefiles in **CASA**. They need not be in a new release of **CASA**. Files ending with `.m` are not listed because these are generated files.

The string `$CASA` should be replaced by the directory where **CASA** resides.

`mgrobner`**:** The following files under `$CASA/mgrobner` are not used by the makefiles.

```
help/mgrobner/src/mgbasis
help/mgrobner/src/mgbasisx
help/mgrobner/src/mnormalf
help/mgrobner/src/msolveGB
help/mgrobner/src/msolveSP
help/text/mgrobner
```

`.comglobal`**:** The file `$CASA/src/.comglobal` has the following contents:

```
/^global.*;[ \t]*$/s/^/#/
/^global/,/;[ \t]*$/s/^/#/
/# protect/s/# //
```

These are some expressions for the Unix program `sed`. This file was probably only useful to transform **CASA** code for **Maple** V.3 to code which also worked under **Maple** V.2 (cf. `$CASA/Makefile`).

Library: The `$CASA/lib` directory is generated.

Help files: The `$CASA/Makefile` contains old code for creation of online help. The directories `text` and `src` under `$CASA/help/casa` were probably used for Release 3 but become useless by the new **Maple** worksheet format. Those files reside in `$CASA/help/casa/mws`. Files not ending with `.mws` in this directory are not used. It even seems to be possible to generate the file in the old help format from the LaTeX files and the script `mkmaplehelp` (which involves the files `refman.el`, `mklm`, `cnv2pp`, and `tpp` under `$CASA/doc` (cf. section 2.6.4).

The file `$CASA/help/text/casa.m` contains an old version of the 'Welcome to CASA' message and seems not to be used anymore.

If **Maple** V.3 is no longer supported, only the files ending in `.mws` must be kept.

Manual files: The creation of the manual uses files in the directory `$CASA/doc`. All files having `_pp` in their name as well as the file `rmLaTeXsrc` are generated by the `doman` perl script.

Except `manual.tex` and `manual.cls`, all files matching `manual.*` are generated and can be deleted.

The files under `$CASA/doc/mgrobner` are not used for the **CASA** manual. They form a separate document which describe the project where the `mgrobner` package was programmed.

The scripts `doman`, `cnv2pp`, `tpp`, and `mklm` can be removed if (1) TEX preprocessing is not involved in the creation of the manual, and (2) the creation of help files in the **Maple** V.3 help style from the `.tex` files is no longer needed.

Source files: Under `$CASA/src` the following files are not used.

```
curves/newrealp
curves/private/myremfactors
curves/private/newsimple
curves/private/pencil.old
curves/private/simple.old
curves/private/tmp
curves/ratPar.old
curves/realRepar
curves/sendra
geoprover/kapur
geoprover/nam
geoprover/nam2
geoprover/nam3
geoprover/polyop
geoprover/polyop1
geoprover/polyop2
grobner/GWalkMinimal
grobner/private
misc/tsolvet
oldStuff/.files.gz
oldStuff/.procs.gz
oldStuff/irreduciblep
plot/pacPlotTransform
plot/pacplot-060597
plot/pacplot.270497
plot/pacplot.morecrit
plot/pacplot.save
plot/pacplot.tmp
plot/private/ChangeLog
plot/ssiPlot.save
plot/texput.log
resultant/#MultiCra#
```

```
resultant/DetInterp
resultant/Inv
resultant/MultiCra
resultant/MultiInterp
resultant/asspr
resultant/cra
resultant/detInterpMod
resultant/dixon-abi
resultant/dixon-asc
resultant/dixon-g
resultant/dixon-lex
resultant/dixon-plex
resultant/dixon-tdeg
resultant/dixon.tmp
resultant/ipcra
resultant/maipdm
resultant/mdinv
resultant/midcra
resultant/mmpdma
resultant/mpint
```

2.2 Conventions

This section concentrates on some conventions concerning the **CASA** code including procedure names, coding standards and error handling.

2.2.1 Procedure Names

Procedures are usually referenced to by calling them with their "long name", i.e. in the format

`casa/`*directoryname*`/`*procedurename*`(`*arguments*`).

Example (long name): `'casa/basics/addattr'(...)`.

An alternative is to use the corresponding "short name" that corresponds in nearly all cases to the *procedurename* of the "long name" (the only exception w.r.t. this convention is `Attributes`, the short name for `casa/basics/attributes`). This "short name" is defined in a table set up by the file `casa/init/casa`, or it is defined by the `casa/init/init` file via the `readlib` command.

Example (short name): `addattr`.

Some words on the choice of names: they often contain abbreviations which reduces readability.

Example (typical **CASA** name): `mkImplAlgSet` for "Make implicit algebraic set".

There is a tendency to capitalize the different words within a name, but this approach is not systematically kept through.

Example (deviation from convention): `projalgpoly` for "projective algebraic polynomial".

2.2.2 Coding Standards

There exist some standards in the **CASA** code concerning the appearance of headers, though they are not strictly kept through all procedures. There exist some deviations from the below given schemes (some procedures even have only a degenerated or no header). We give some example headers that are chosen such that each one represents a scheme for a larger class of occurring headers.

Example 1:

```
## -*- maple -*-
#######################################
#   Algebraic Set Package RISC Linz
#######################################
#   File:     attributes
#   Content:  representation of algebraic sets
#   Author:   B. Wall
#   Date:     July 90
#   Last Updated: 1st April 1996 by T.Q.Nam
```

Example 2:

```
## -*- maple -*-
########################################
#   Algebraic Set Package RISC Linz
########################################
#   Author:   B. Wall
#   Date:     July 90
#   Last Updated: March 18, 1996 by T.Q.Nam
```

Example 3:

```
##############
#    CASA    #
##############

#  Name : shAlgSet
# Author: Tran Quoc-Nam
#  Date : 26/06/96

# Context :
# Package Version: CASA 2.3
# Copyright: RISC-Linz 1996
```

```
# History: B. Wall, July 90
# Keywords: Now we can manipulate the truncated power series.
# Limitation:
```

Example 4:

```
## -*- maple -*-
##
##    Title:    assign
##    Created:  Mon Nov 29 15:38:34 1993
##    Author:   Michael Mnuk
##              <mmnuk@melmac>
##
## Description: function 'assign' returning the assigned va-
##              lue
```

Some uniform scheme for headers would be preferable.

The code is insufficiently documented, usually a procedure contains no comments at all.

The basic rules of indentation for structuring procedures, their control structures, and the like are usually followed within each procedure, though no standards seem to exist that would guarantee some uniform appearance of the code.

2.2.3 Error Handling

Usually, all top-level procedures in **CASA** test in some way that their input arguments are of the assumed type, while auxiliary functions usually rely on the fact that they are called with appropriate arguments. In **Maple** V.4, a kind of type declaration within the parameter list is possible; it is of the form:

procedurename(...,*parameter*::*type*,...).

Parts of the more recent **CASA** code make use of this facility. But for cases where more refined type checking is necessary, or where incompatibilities are only detected at some point within the procedure, another way of error handling is chosen: the procedure is aborted by calling **Maple**'s `ERROR` function, thereby giving some hints to the user about what error might have occurred. The `ERROR` command has the following syntax:

`ERROR`(*sequence*);

Maple then prints the following error message:

Error, (in *procedurename*), *sequence*

Errors that are returned by invoked procedures sometimes are handled via **Maple**'s `traperror` command. A scheme of how this can be achieved might look like this one:

variablename := `traperror`(*procedurename*(...));
if *variablename* = `lasterror` then `ERROR`(`lasterror`);

Here is a list of **CASA** procedures in which the command `lasterror` occurs (names of files are as usual given relative to the installation directory):

```
convert/impl2plac.m
convert/impl2plac
curves/ratPar.m
geoprover/geoprove.m
init/init.m
init/init
init/casa.m
init/casa
misc/pointInAlgSet.m
misc/pointInAlgSet
plot/ssiPlot.save
plot/ssiPlot.m
plot/ssiPlot
plot/pacplot.morecrit
plot/pacplot.270497
plot/pacplot-060597
plot/pacPlotTransform
```

2.3 Types in CASA

In this section the types which are defined in the **CASA** system will be described.

2.3.1 Description of the Algebraic Set Type

An algebraic set can be represented in four different forms.

Implicit representation: An algebraic set is the set of common zeros of a system of polynomial equations. To give an algebraic set in implicit form means to give finitely many polynomials.

Projected representation: An algebraic set in projected form is given by a set of polynomials, describing an implicitly given algebraic set, and a tuple of rational functions (specifying a birational mapping into a space of higher dimension).

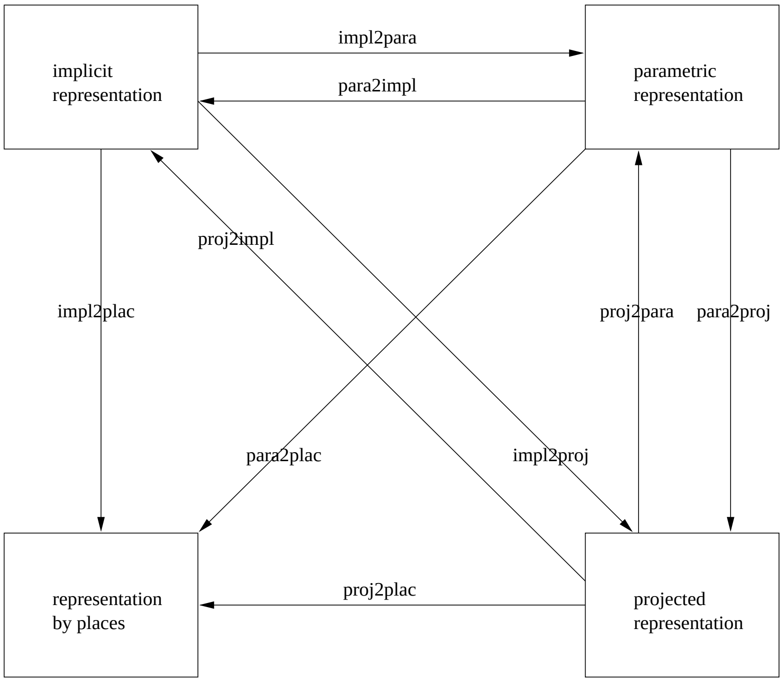
As a consequence of the primitive element theorem every irreducible d-dimensional algebraic set in n-dimensional space is, after a suitable linear transformation of coordinates, birationally projectable onto an irreducible d-dimensional algebraic set in $(d+1)$-dimensional space, which can be specified by a single polynomial in $d+1$ variables. This can be generalized to unmixed-dimensional algebraic sets.

The projected representation in **CASA** is mainly used for space curves. They are described by an algebraic set given by one polynomial in two variables and a birational map which maps this algebraic set into a higher dimensional space.

Parametric representation: Some irreducible algebraic sets can be parametrized by rational functions. In the case of curves these are exactly the curves of genus zero. An algebraic set in parametric form is given by a tuple of rational functions that parametrizes the algebraic set.

Representation by places: All algebraic curves can be locally parametrized by a set of power series that are convergent around a point of the curve. An algebraic set is given locally by places if for each branch passing through a certain point on the algebraic set a tuple of power series that parametrizes the algebraic set around the point is specified.

Different representations can be converted into each other. The available conversions in **CASA** can be read off from the following graph.



There are nine conversion functions. However, even when there is a conversion routine from one representation to another, it may not be applicable to all algebraic sets in the input representation. In general, **CASA** is mainly designed to handle curves, thus the conversion of non-curves is only available (with some limitations) between implicit and projected representation and from a parametric to implicit and projected representation.

Each conversion function checks whether its input parameter is an algebraic set in the right representation. If this is not the case, the function aborts with an appropriate error message. In the sequel we will tacitly assume that all the algebraic sets are given in appropriate representation.

Usually, algebraic sets will be input in implicit or parametric form by a user. All algebraic sets can be given in implicit form. The representation by places describes the local behavior of a curve around a point. So the representation by places depends on such a point. An algebraic set in projected form always describes an algebraic set with components of the same dimension. Therefore a conversion of an algebraic set of mixed-dimensional components to a projected form will result in not only one but several algebraic sets in projected representation.

`impl2para`: The function call to

```
impl2para(a)
```

roughly returns

```
proj2para(impl2proj(a)),
```

so the restrictions of those two functions apply.

`impl2plac`: If no point is given as a second parameter, the center of the representation by places defaults to the origin of the coordinate system, otherwise it is the given point.

The function calls `impl2proj`. If the algebraic set is mixed dimensional the function aborts with showing the sequence of unmixed-dimensional components and asks the user to call `impl2plac` on each component in turn.

If the given point is not on the curve, the function exits with an error message.

If there are two points of the projected form of the given algebraic set, which map to the center, the function aborts with the error message

```
Choose different projection direct - ambiguous projected center
```

which does not really tell the user what to do in such a case.

Then the function `proj2plac` is called with the algebraic set obtained from the previous `impl2proj` and the corresponding center.

The conversion is done by means of Duval's version of the Newton polygon method.

`impl2proj`: If the given algebraic set is empty the procedure aborts with an error message, otherwise it is decomposed into unmixed-dimensional algebraic sets. Then for each of these sets a projected form is computed.

Optionally a second parameter, a variable list, can be given. The first variable name will be taken to rename the dependent variable and the others to rename the independent variables in the resulting algebraic set(s). An error occurs if this list is shorter than $d+1$ where d is the dimension of the given algebraic set.

`para2impl`: The function applies to an algebraic set of arbitrary dimension and to an optional second parameter which must be a list of variable names that are used for the polynomials of the implicit representation. If the second parameter is not provided, it defaults to

```
['x', 'y', 'z']
```

i.e. the target space is the 3-dimensional affine space.

An algebraic set in parametric form is represented by a list of rational functions. The length of the list corresponds to the dimension of the affine space where this algebraic set is embedded. The length of the list of the second parameter must be equal to this dimension; only in the case where this dimension is 1 or 2, the length of the variable list (second parameter) can be greater or less, since otherwise the procedure aborts with an error (the variable list is too short) or produces wrong results as the following example shows:

```
A:=mkParaAlgSet([0,s,t],[s,t]);
```

$$\mathrm{algebraic\_set}([0, s, t], [s, t])$$

```
B:=para2impl(A,[a,b,c,d]);
```

$$\mathrm{algebraic\_set}([a], [a, b, c, d])$$

The dimension of A is 2 whereas the dimension of B is 3.

`para2plac`: This function only works only for curves in parametric form. As an optional second parameter one could give a point, the center of the Puiseux expansion (defaults to the origin if not explicitly given).

The function aborts if the given point does not lie on the curve.

As an optional third parameter a list of variables can be given that is used as names in the Puiseux expansion. The function aborts if this parameter is not a list of names.

`para2proj`: The function applies to algebraic sets of arbitrary dimension. The transformation from parametric to projected form is based on the fact that the parametric representation is a special case of the projected representation. An algebraic set in parametric form can be seen as being in projected form where the hypersurface is a hyperplane. It is the hyperplane defined by x=0 where x is a variable not occurring in the rational functions that constitute the parametrization (and the birational mapping).

Therefore, the length of the list of variable names, which can optionally be given as a second parameter, should contain one variable more than the variable list of the parametric representation. If this second parameter is not given, the variables will be taken from the parametric representation and a new variable `_ZZ` will be added. Otherwise the second parameter will be checked to be of correct length, and the function aborts if this is not the case. It is, however, not checked whether or not this parameter is, in fact, a list of names.

`proj2impl`: The function applies to algebraic sets of arbitrary dimension. The function can be invoked with an optionally second argument, a list of variables in which the polynomials of the resulting implicit form should be given. The number of these variables should coincide with the number of rational functions in the projected representation of the given algebraic set. If the variable list is shorter, some variables will be added automatically. If the list is longer, the function aborts with an error message.

`proj2para`: The given algebraic set must represent an irreducible curve given by exactly one equation. If this is not the case the function aborts with an error message.

`proj2plac`: The function applies to a curve given in projected form by exactly one polynomial in two variables.

As an optional second parameter one can give a point, the center of the Puiseux expansion, which defaults to the origin if not present.

It seems not to be checked whether or not the given point is indeed a point on the curve.

For **CASA**, the variable `algebraic_set` should not have a value. An algebraic set is represented by an expression of the form

```
algebraic_set(def,vars,attributes)
```

where the fields have the following interpretation:

`def`: The type of this field depends on the value of the field rep of the table that is given by `attributes` (see below). According to the representation type, we have the following.

Implicit representation: list of polynomials

Parametric representation: list of rational functions

Representation by places: list of places where each place is a list of power series stored as procedures

Projected representation: list with 2 entries

1. list of polynomials
2. list of rational functions

It is unclear why the representation allows lists of polynomials when the idea is to represent an irreducible d-dimensional algebraic set by a hypersurface (i. e., one polynomial) in $(d+1)$-space.

`vars`: A list of variable names. Other variables occurring in the polynomials `poly` are taken to be parameters.

`attributes:` table with two entries

- **rep:** Stores one of `impl`, `para`, `proj`, or `plac`, the type of the representation of this particular algebraic set. This field shows whether the algebraic set is given in implicit, parametric, or projected representation, or in a representation by places.
- **oth:** This is a list of `key=value` pairs, the attribute list that will be updated by `addattr`. For the possible keys and their meanings see below.

Constructors An instance of the type `algset` (algebraic set) is created by one of the functions:

`casa/basics/mkImplAlgSet`,
`casa/basics/mkParaAlgSet`,
`casa/basics/mkPlacAlgSet`, and
`casa/basics/mkProjAlgSet`.

Selectors The following functions extract information from an algebraic set:

`casa/basics/attributes`,
`casa/basics/hasattr`,
`casa/basics/hasattrib`,
`casa/basics/repattr`,
`casa/basics/represent`, and
`casa/basics/varlist`.

Information The function `casa/basics/shAlgSet` prints all known information about an algebraic set, including representation, variables and attributes.

Printing The function `print/algebraic_set` is used whenever **Maple** needs to display an algebraic set.

Type Checking The function `type/algset` is called implicitly by **Maple** in expressions like:

```
`casa/basics/hasattrib`:=proc (att, a::algset) ... end
```

or

```
type(a,algset).
```

Attribute Setting The function `casa/basics/addattr` operates destructively on its second argument, an algebraic set. The calling sequence is:

```
addattr(key=value,a).
```

The `key=value` pairs will be added to the attribute list of the algebraic set `a` without checking whether the `value` is consistent to the actual properties of the involved algebraic set. Usually, properties of an algebraic set will be added automatically by the respective procedures like, for example, `genus` or `dimension`, in order to avoid expensive computations when they are called for the second time. It may however happen that some information is known to the user before. For example, it may be known in advance that the dimension of the algebraic set is zero. Therefore, one could add the property `dimension=0` to the algebraic set. The responsibility to add only consistent attributes is left to the user. Note that it would be possible to add `dimension=1` for a zero-dimensional algebraic set; this, however, could later lead to wrong results since the **CASA** code relies on the correctness of the information stored in the attribute list. So the function `dimension` would not compute the correct value of the dimension and wrongly return 1 as an answer.

Keys There exist the following keys:

basespace: The corresponding value can either be `affine` or `projective`. Usually, this should be already given in the creation of the algebraic set through the `mk`*xxxx*`AlgSet` functions. If it is omitted, an affine space is assumed. We do not see a reason to use `basespace` in `addattr`, and, in fact, it can lead to inconsistencies as can be seen from the results of the following code example.

```
A:=mkImplAlgSet([x^2+y^2+z^2],[x,y,z]);
B:=mkImplAlgSet([x^2+y^2+z^2],[x,y,z],basespace=projective);
dimension(A);
dimension(B);
addattr(basespace=projective,A);
dimension(A);
```

Initially A has dimension 2 and B has dimension 1. Since the dimension is stored in the property list of A, the second call to dimension will wrongly report 2.

The key `basespace` appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/mkImplAlgSet
casa/basics/mkParaAlgSet
casa/basics/mkPlacAlgSet
casa/basics/mkProjAlgSet
casa/convert/convertSpace
casa/curves/passGenCurve
casa/curves/ratPar
```

```
casa/dimension/dimension
casa/init
casa/misc/tangSpace
```

dimension: The dimension of the algebraic set.

This key appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/mkImplAlgSet
casa/basics/mkParaAlgSet
casa/basics/mkPlacAlgSet
casa/basics/mkProjAlgSet
casa/basics/shAlgSet
casa/dimension/dimension
casa/dimension/independentVariables
casa/plot/plotAlgSet
```

genus: The genus of the curve represented by the algebraic set.

This key appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/mkImplAlgSet
casa/basics/mkParaAlgSet
casa/basics/mkPlacAlgSet
casa/basics/mkProjAlgSet
casa/basics/shAlgSet
casa/convert/proj2para
casa/convert/proj2paraOld
casa/curves/genus
```

groebnerbasis: The value is a list with two entries:

1. a term order,
2. a Gröbner basis.

The value makes only sense for an algebraic set in implicit representation. Then it is the Gröbner basis of the defining polynomials of the algebraic set with respect to the corresponding term order.

This key appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/mkImplAlgSet
casa/basics/mkParaAlgSet
casa/basics/mkPlacAlgSet
casa/basics/mkProjAlgSet
casa/basics/shAlgSet
casa/dimension/private/dimensionimpl
casa/misc/Groebnerbasis
casa/setops/implSubSet
```

indepvars: Stores the independent variables of an algebraic set given in implicit representation which are computed during the computation of the dimension of the algebraic set.

This key appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/shAlgSet
casa/dimension/dimension
casa/dimension/independentVariables
```

proper: Stores a proper parametrization of an algebraic set in parametric form.

This key appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/shAlgSet
casa/misc/properParametrization
casa/plot/plotAlgSet
```

showTermsOfExpansion: Number of terms of the power series that should be shown by the procedure `shAlgSet` when displaying an algebraic set in representation by places.

This key appears explicitly in the procedures:

```
casa/basics/addattr
casa/basics/shAlgSet
casa/convert/proj2plac
```

Also the keys

`implrep`,
`pararep`,
`projrep`,
`isprojective`,
`neighborhood`,
`irreducible`, and
`reducible`

are allowed, but only the procedures `addattr`, `mkImplAlgSet`, `mkParaAlgSet`, `mkPlacAlgSet`, and `mkProjAlgSet` use them and so they seem to be unimportant in **CASA**.

2.3.2 Description of the Type `casaCoeffType`

The type `casaCoeffType` is one of `rational`, `algnum`, `algnumext`. Each of the last three types are standard **Maple** types representing the rational numbers, algebraic numbers, or extensions of algebraic numbers. The **Maple** online help says:

> An algebraic number is defined as either a rational number or a root of a univariate polynomial with algebraic number coefficients,

specified by a `RootOf`. A sum, product, or quotient of these is also an algebraic number.

An algebraic number extension is a root of a univariate polynomial with algebraic number coefficients, specified by a `RootOf`.

2.3.3 Description of the Type `projalgpoly`

There is a type checking function `type/projalgpoly`, but no procedure to create such objects of this type.

It will be checked whether the first parameter is a homogeneous polynomial in the variables that are given as a second parameter over a coefficient type that depends on the third parameter. If the third parameter is the string `system` then the coefficients are taken to be polynomials in the remaining variables over `casaCoeffType`, otherwise the coefficient must just be of `casaCoeffType`.

2.3.4 Description of the Type `planecurve`

There is no creating procedure, only a type checking function `type/planecurve` exists.

A `planecurve` is an algebraic set in implicit representation. The corresponding list of polynomials should contain only one element, which is either bivariate or homogeneous in three variables.

This procedure seems to have a bug, since the call

```
type(mkImplAlgSet(x,[x,y,z,u]),planecurve)
```

reports also true.

2.3.5 Description of the Type `projpoint`

An object of this type is a non-empty list of elements of type `casaCoeffType` where at least one entry is not equal to zero.

2.3.6 Description of the Type `EnhRootOf`

The function `type/EnhRootOf` was introduced as a workaround to some problems in testing for `RootOf` expressions with **Maple**'s library `` `type/RootOf` `` procedure. This functions tests whether its argument is of the form

```
RootOf(x)
```

for some expression x. There are no test files which show the problem, even the author of this procedure (M. Mňuk) does not have such files. It is, therefore, unclear whether the procedure `type/EnhRootOf` can be discarded due to a possible bug fixing in new **Maple** releases.

2.4 Package Dependencies

On the next page we find the package dependencies visualized via the Package Dependence Graph. An edge in this directed graph from one directory vertex `dir1` to another directory vertex `dir2` exists if and only if some procedure in `dir1` calls some procedure in `dir2`. The names of the vertices in this graph reflect the corresponding directory names.

Immediately visible from the graph is that the packages `grobner` and `extnewton` are completely independent from the rest of **CASA**, and that `realroot` and `resultant` are only called by `plot`. Others like `basics`, `misc`, and `convert` are among the most used packages.

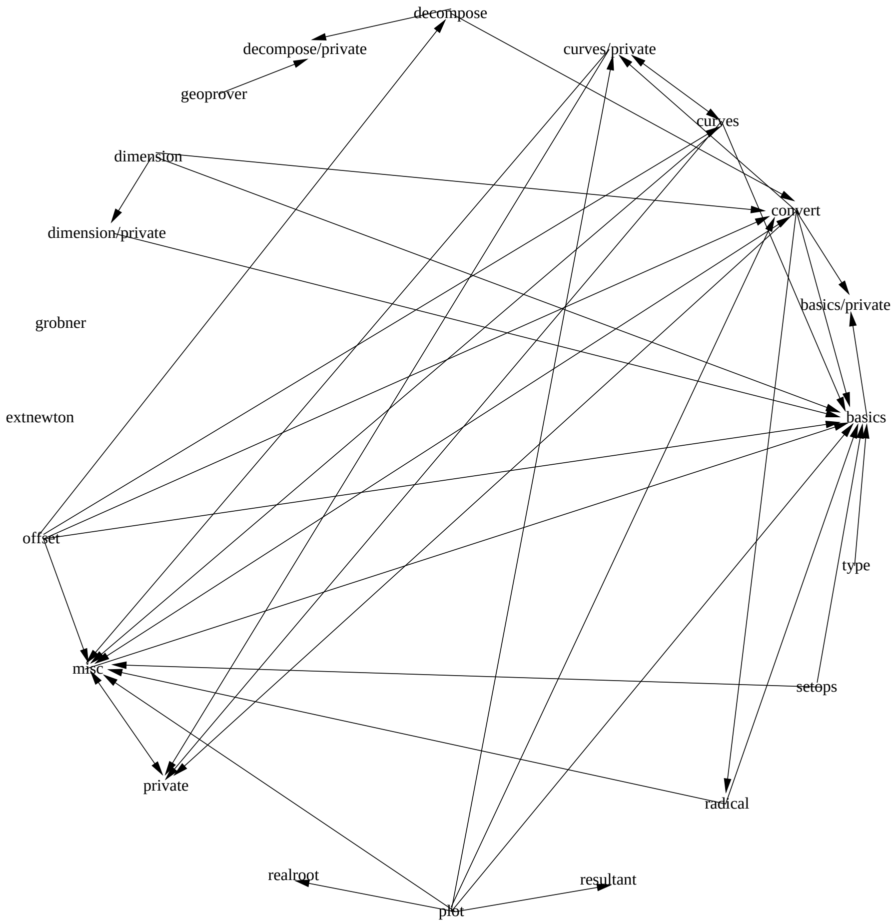
Another notable point may be that some *directoryname*/`private` subdirectories are also called from directories different from their mother directory, which somehow contradicts the intended "privacy" of those subdirectories.

The role of the package `type` is not visible in the Package Dependence Graph, since the latter one is based exclusively on the analysis of procedure calls, which does not include type checking.

A division of the totality of all packages into three categories might look like this:

Low-Level Pack.	Medium-Level Pack.	Application Pack.
`basics`	`curves`	`dimension`
`convert`	`decompose`	`extnewton`
`misc`	`radical`	`geoprover`
`type`	`realroot`	`grobner`
	`resultant`	`offset`
		`plot`
		`setop`

By low-level packages we understand those packages that are extensively used by many other packages. Application packages are those that mainly use other packages. Finally, packages that do not fall into one of the above two categories are classified as medium-level packages.



2.5 CASA Start-up

2.5.1 Instructions for the CASA User

In order to start **CASA** one has to set the **Maple** variable `libname` to point to the path where **CASA** can be found.

Currently, the library files for **CASA** at RISC are in the directory

`/home/alggeo/casa-2.3/lib`.

We use this path here as an example.

One should issue the command

```
libname:=`/home/alggeo/casa-2.3/lib`,libname:
```

in **Maple** or put this line into the file called `.mapleinit` in one's home directory. It is not yet clear, how the automated setting via an initialization file works under Windows.

If this is done, one can issue the command

```
with(casa);
```

to load **CASA** into the **Maple** system.

At this moment **Maple** will look for a definition of a table with the name `casa`. Since we prepended the path to **CASA** to the `libname`, **Maple** will first look for this table in the library

```
/home/algeo/casa-2.3/lib/maple.lib
```

where this table is, in fact, defined.

2.5.2 Information for the CASA Maintainer

The following subsection describes what happens when **CASA** is loaded into a **Maple** session.

In the following, the string `$CASA` has to be replaced by the directory where **CASA** resides. At the moment **CASA** at RISC is located in the directory `/home/algeo/casa-2.3`.

The `casa` table

CASA is a **Maple** package and, therefore, must define a table which contains all exported **CASA** functions.

The corresponding source file is located in

```
$CASA/src/init/casa.
```

The table `casa` in this file consists of a collection of

```
name = procedure
```

pairs which make according to the **Maple** philosophy (cf. [2]) short names available to call **CASA** functions.

The line

```
addattr = proc() `casa/basics/addattr`(args) end,
```

is an example of a real entry in this table.

The entries in this table `casa` are grouped into the following sections.

- initialization of **CASA**
- basics

- convert
- radical
- dimension
- setops
- plot
- geoprover
- mgrobner package
- grobner
- extNewton
- realroot
- resultant
- misc
- decompose
- curves
- offset
- private

Except for `mgrobner` these sections correspond to subdirectories in \$CASA/src. The section mgrobner corresponds to the directory `$CASA/mgrobner`.

In `$CASA/src` exist two other directories, namely `oldStuff` and `type`.

The Initialization Procedure

Since this table `casa` contains as the first entry a definition of an initialization function, the corresponding procedure (`casa/init`) is called at the time the `with(casa)` command is executed.

The main task of the initialization procedure is to protect the short names of **CASA** functions and to define some global variables and types for the use in **CASA**. A user can provide a function with the name `casa/user-init`, which will be called at the end of the initialization procedure.

Since the function `casa/init` defines global variables with the same name as the short names in the table, the table will actually never be used. The `with(casa)` command first creates global variables with the name of the lhs of a table entry and assigns then the rhs. After this is done, the function `casa/init` will be called, which immediately overwrites the assigned values by some unevaluated `readlib` commands.

This initialization function checks whether a global variable `casa/loaded` was already defined and if so does not execute the function `casa/init`. Then the variable `casa/loaded` is set to `true` and the function `casa/init` is called, which is defined in the file `$CASA/src/init/init`.

The intention of this file is not yet quite clear since it makes all(?) identifiers which are defined in **CASA** global and later protects them. The main idea probably is to protect **CASA** function names from accidental overwriting by some user and to provide some predefined variables as, for example, `casa/homvars`.

The structure of the file is the following:

1. The whole file contains the definition of the procedure `casa/init`.
2. The variable `a` is declared to be local. This variable is later assigned to a possibly user-defined function `casa/user-init` and executed.
3. The copyright notice with the drawing of the castle appears.
4. Exported **CASA** variable names (short and long form) are declared to be global.
5. The options part contains the name `CASA`. What this is used for is not yet clear.
6. The `casa` table is extended by two entries, namely

   ```
   casa[CoeffType]:=`type/casaCoeffType`:
   casa[Homvars]:=`casa/homvars`:
   ```

7. If a `casa/user-init` evaluates to a procedure or a name of a procedure which can be loaded via `readlib`, such a procedure is executed without any arguments.
8. A number of definitions of varibles via `readlib` commands, for example,

   ```
   addattr    := 'readlib('`casa/basics/addattr`')':
   Attributes := 'readlib('`casa/basics/attributes`')':
   ```

9. The following `readlib` commands occur without an assignment.

   ```
   readlib(`casa/curves/private/sortalg`);
   readlib(`casa/curves/private/sortfac`);
   readlib(`casa/curves/private/sortrat`);
   readlib(`casa/curves/private/sortsings`);
   readlib(`casa/geoprover/dictionary`):
   ```

10. The following variables are assigned too.

    ```
    `casa/homvars`:=[x,y,z]:
    `type/casaCoeffType`:={rational,algnum,algnumext}:
    ```

11. A list of reserved symbols is defined. It contains short forms of names of types in **CASA** as well as the short forms of procedure names.

    Apart from **CASA** procedure names, the following symols are listed:

    - `t`,
    - `T`,
    - `caso_raro`,
    - `non_egular`,
    - `unlucky`,
    - `exito`,
    - `basespace`,
    - `projective`,
    - `affine`,
    - `algebraic_set`,
    - `impl`,
    - `para`,
    - `fini`,
    - `proj`,
    - `implrep`,
    - `pararep`,
    - `projrep`,
    - `rep`,
    - `algset`

12. By the command `map(protect,‘casa/ReservedSymbols‘)` the symbols above are protected from accidental overwriting.

13. Now some help related `readlib` commands occur. They seem to be out of date, because the corresponding files are missing.

Miscellaneous

The package `mgrobner` is treated as a **Maple** package: no command of this package is listed.

The command `casa/misc/assign` does not have a short name.

Convention with the `readlib` Command

The `readlib` command is used to delay the loading of the actual code. In this way code will be loaded if the function is called for the first time.

There always appear pairs of assignments of the following form.

```
addattr := 'readlib(''casa/basics/addattr'')':
`casa/basics/attributes`:='readlib(''casa/basics/attributes'')':
```

If the function `addattr` is used for the first time, the `readlib` command will be evaluated. By this the function `` `casa/basics/attributes` `` is loaded from the library. As a side effect the variable with the same name gets the value from the library (which is a procedure) assigned.

The value of `addattr` is not changed. It remains an unevaluated `readlib` command. It is not clear whether **Maple** remembers the fact that the function `` `casa/basics/attributes` `` was already loaded and does not need to access the library a second time.

In case the variable `` `casa/basics/attributes` `` is used for the first time, the same as above happens except that a second call to the function does not involve the `readlib` command anymore, because the side effect changed the initial value of the variable `` `casa/basics/attributes` ``.

2.6 How to Make CASA?

The following sections apply to the creation of **CASA** under Unix. Because **Maple** V release 4 was not available under other operating systems, it could not be checked whether the following also applies to other operating systems. However, because of the usage of many Unix features, it seems likely that only Unix can be used to build everything of **CASA**.

The string `$CASA` has to be replaced by the directory where **CASA** resides, for example, `/home/alggeo/CASA` (no slash at the end).

2.6.1 Making the Library

1. Make sure the directory `$CASA/lib` exists. This is where the compiled library will go.
2. Go to the directory `$CASA/src` and type

   ```
   make casa
   ```

   to compile all the **CASA** source files and to create the library files

   `maple.lib` and `maple.ind`

   under the directory `$CASA/lib`.

2.6.2 Making the User Manual

Since the command

```
make manual
```

does not work properly, one has to issue all the steps by hand.

The corresponding entry in `$CASA/doc/GNUmakefile` looks like this:

```
doman -LaTeX
latex manual_pp
bibtex manual_pp
xon jade makeindex $(CASATOP)/doc/manual_pp.idx
latex manual_pp
latex manual_pp
rm -f manual.dvi
ln -s manual_pp.dvi manual.dvi
dvips -o manual.ps manual.dvi
```

The printed manual, however, seems to be produced without using the TEX preprocessor. This can be done by the following commands:

```
latex manual
bibtex manual
makeindex manual
latex manual
latex manual
dvips manual
```

invoked in the directory `$CASA/doc`. The files `manual.dvi` and `manual.ps` will be produced.

The scripts `doman`, `cnv2pp` `tpp`, and `mklm` will no longer be used, because TEX preprocessing is not involved.

2.6.3 Making the WWW Pages

1. Change to the directory `$CASA/doc`.
2. Save the file `plotAlgSet.tex`.

   ```
   cp plotAlgSet.tex plotAlgSet.tex.save
   ```

3. Change line 45 from

   ```
   \begin{tabular}[t]{lp{11.5cm}}
   ```

   to

   ```
   \begin{tabular}[t]{ll}
   ```

4. Change to the directory `$CASA/www`.
5. Make sure that an older emacs is used for hyperlatex. (emacs-19.34.1 works fine.) Let the `PATH` variable point to that emacs.
6. Issue the command

   /usr/local/bin/hyperlatex-1.4 casa.tex

This will create the files `casa.html`, `casa_1.html` up to `casa_4.html`.

7. Copy those files to the place where the web pages should go, for example, `/home/info/www/software/casa`.
8. Copy also the files with the extension `.GIF` and `.ps.gz` and the file `index` to the web directory. Make sure that the postscript files are up-to-date versions.
9. Change to the web directory. Make sure all files and directories have the appropriate modes.
10. Issue the command `wndex` in the web directory.
11. Change to the directory `$CASA/doc`.
12. Restore the file `plotAlgSet.tex`.

    ```
    mv plotAlgSet.tex.save plotAlgSet.tex
    ```

2.6.4 Making Online Help

1. Change to the directory `$CASA`.
2. Change the variable `CASATOP` in the file `help.maple` to the directory where **CASA** resides. This should be the current directory, for example, `/home/alggeo/CASA/`. The slash has to be there.
3. Start an **XMaple** session.
4. Enter the following **Maple** command:

    ```
    read(`help.maple`);
    ```

    This will create the file `$CASA/lib/maple.hdb`.
5. Leave **XMaple**.

Files in the old help style format (**Maple** V.3) can be generated as follows:

1. Change to the directory `$CASA/doc`.
2. Edit the file `mkmaplehelp`.
3. Change line 10 from

    ```
    $mapledir="/home/alggeo/work/doc";
    ```

    to the directory where the generated file should go.
4. Change the `load-path` in line 30 to the current directory.
5. Save and quit editing `mkmaplehelp`.
6. For each `.tex` file execute the command

    ```
    mkmaplehelp filename.tex
    ```

    A file *filename* will be created in the directory which was entered in line 10 of `mkmaplehelp`.

2.7 Requirements for Installation and Use

In order to run the current version of **CASA** (2.3), one needs a computing environment that provides **Maple** V.4.

2.7.1 Installation of the System

As **CASA** is distributed as a **Maple** library, the installation basically amounts to just copying files into appropriate places. Thus it should be possible to install **CASA** without any problems. Details about the installation procedure can be found in the "`INSTALL`"-file.

2.7.2 Documentation for the User

There are three main sources of documentation and help for the user.

Web pages: The **CASA** web page with an introduction to **CASA** and a collection of function descriptions is located at

`http://www.risc.uni-linz.ac.at/software/casa`.

Help-files: There are **Maple** help-files (with `.mws` extension) for an online documentation. In order to get help for a specific command, one has to type

?*command*

within the **Maple** environment.

Handbook: The **CASA** Reference Manual [9] (written in LaTeX) is available. It contains an introduction to **CASA**, an overview of new features, a chapter on function descriptions and an appendix on writing documentation.

2.7.3 Documentation for the Programmer

There seems to be no documentation for programmers or maintainers. Desirable would be documentation related to topics exhibited by the following questions:

- How to compile **CASA**?
- How to create documentation?
- How to write a **CASA** package?

Chapter 3

Description of Target State

In this chapter we outline some goals for improving the **CASA** system with respect to installation, portability and expandability in the future. These goals are based on the analysis of the current state of **CASA** as described in Chapter 2. A coarse structuring of these goals encompasses three different areas:

1. goals concerning the software,
2. goals concerning the documentation,
3. long-term goals.

3.1 Goals Concerning the Software

In this section we outline the goals for improving the software of the **CASA** system. We identified the following eight goals:

1. erasing unused files,
2. simplification of the installation procedure,
3. writing new versions of the initialization files,
4. reorganization of sub-packages,
5. coherent use of long names for procedures,
6. standardizing headers and interface descriptions,
7. standardizing type handling,
8. introducing revision control.

3.1.1 Erasing Unused files

Problem: There exist several files that are not used by any **CASA** `makefile` (see Subsection 2.1.2 ("Unused files") for a list of those files).

Solution: These superfluous files have to be erased.

3.1.2 Simplification of the Installation Procedure

Problem: The current installation procedure involves creating new directories, copying files, setting paths, ... These tasks could be simplified by a higher level of automation.

Solution: Tools that automate the above mentioned tasks and hence the installation procedure have to be created.

Work to be done: Provide a binary and a source release. The binary release should only contain the precompiled library and a file that tells the user how to make **CASA** working on his platform (Unix or Windows NT).

The source release should only contain files that are not generated, together with those scripts and makefiles that generate the **CASA** library and the various kinds of documentation.

3.1.3 Writing New Versions of the Initialization Files

Problem: There are two cases of **CASA** procedures, where the consistent allocation of a short procedure name would correspond to an existing **Maple** procedure name. The two deviations are:

1. the procedure `casa/assign` has no corresponding short name.
2. the short name of the procedure `casa/attributes` is `Attributes`.

Solution: Both procedure names have to be capitalized in their short version. They have to be exported, i.e. they should be visible for the user.

Work to be done: The initialization table `$CASA/init/casa` and the initialization file `$CASA/init/init` have to be updated.

3.1.4 Reorganization of Sub-packages

Problem 1: Some packages call procedures in the `private` directory of another package. Namely:

1. `convert` calls `basics/private` and `curves/private`,
2. `plot` calls `curves/private`, and
3. `geoprover` calls `decompose/private`.

Solution 1: The procedures in `private` directories that are called as above described have to be placed one level higher, namely in the mother directory *directoryname* of their directory *directoryname*/`private`.

Problem 2: The `mgrobner` package is not in the `src` directory where all other **CASA** packages are.

Solution 2: There are two possible solutions:

1. the `mgrobner` package might be completely separated from **CASA** and made into a **Maple** package of its own.
2. the `mgrobner` package might be completely embedded into **CASA**; in this case, all short procedure names have to be replaced by their corresponding long name.

Problem 3: Some **CASA** code files contain many local procedures in addition to some top level procedure, which are usually referenced by their short name.

Solution 3: The local procedures should be placed in the corresponding directory `private`, and their short names have to be replaced by long ones. In the `private` directories, it is allowed that one file may contain several procedures.

3.1.5 Coherent Usage of Long Names for Procedures

Problem: The **CASA** code makes use of short and long procedure names in an inconsistent way.

Solution: All short procedure names have to be replaced by their corresponding long name. This is a general task that concerns the **CASA** code as a whole.

3.1.6 Standardizing Headers and Interface Descriptions

Problem: In the current **CASA** code, there is no uniform scheme for headers and interface descriptions.

Solution: Some standard for headers and interface descriptions has to be found and applied for the top level procedures at least.

A suggestion for a header/interface standard might look like this:

```
##################################################################
# CASA: Computer Algebra Software for Algebraic Geometry         #
#                                                                #
# Copyright (C) Research Institute for Symbolic Computation      #
#               RISC Linz, Austria/Europe                        #
#                                                                #
# http://www.risc.uni-linz.ac.at/software/casa                   #
##################################################################
#
# Author: <name (email) of original author(s)>
#
```

```
# <RCS Id String including author and date of last update>
#
# Function: <long name of the function>
#   <Short description of what the function does.>
#
# Calling Sequence:
#   <result> := <fun>(<args>, <a1>, ...)
#
# Parameters:
#   <a1>: <Maple type as specific as possible>
#     <Short description>
#   <a2>: <Maple type as specific as possible>
#     <Short description>
#     ...
#
# Result:
#   <result>: <Maple type as specific as possible>
#     <Short description>
#
# Errors:
#   <Description of when the function aborts with an ERROR>
#
# Synopsis:
#   <Detailed description of the function. How does it work?>
#
# References:
#   <References to the origin of the algorithm.>
#   <Use bibstrings as defined in casa.bib>
##########################################################
```

As a concrete example of a standardized header, take the following.

```
###################################################################
# CASA: Computer Algebra Software for Algebraic Geometry          #
#                                                                 #
# Copyright (C) Research Institute for Symbolic Computation       #
#               RISC Linz, Austria/Europe                         #
#                                                                 #
# http://www.risc.uni-linz.ac.at/software/casa                    #
###################################################################
#
# Author: Klaus Aigner <Klaus.Aigner@risc.uni-linz.ac.at>
#
# $Id: Header.tex,v 1.7 1998/10/13 12:54:55 hemmecke Exp $
#
# Function: GWalk
#   GWalk computes the reduced Groebner basis of the polynomial
#   system F in the indeterminates X1 with respect to the
#   lexicographical term order induced by the order of the
#   indeterminates in X1.
#
# Calling Sequence:
#   G := GWalk(F,X1)
#   G := GWalk(F,X1,X2)
```

```
#
# Parameters:
#   F: list(polynom(C,X1))
#     A list of polynomials in variables X1 over some coefficient
#     domain C.
#   X1: list(name)
#     A list of variable names. The order in this list induced the
#     lexicographical term order for the resulting Groebner basis.
#     The list of indeterminates X1 = [x_1,x_2,...,x_n] induces a
#     lexicographical order where x_1>x_2>...>x_n.
#   X2: list(name)
#     A list of the same variable names as occurring in X1 to
#     specify the order of the variables for the intermediate
#     computation of a total degree Groebner basis.
#     The order of the indeterminates in X2 = [x_1,x_2,...,x_n]
#     induces a total degree reverse lexicographical term order
#     (tdeg) with x_1>x_2>...>x_n.
#     If the function is called with only two arguments, X2
#     defaults to X1.
#
# Result:
#   G: list(polynom(D,X2))
#     A lexicographical Groebner basis of F. The domain D is the
#     quotient field of the domain C.
#
# Errors:
#   The function will be aborted when it is called with less than
#   two arguments.
#
# Description:
#   The function first computes a Groebner basis with respect
#   to the total degree order (tdeg) induced by the order of
#   variables given in X2 by calling the function grobner[gbasis]
#   and converts it to a lexicographical Groebner basis by means
#   of the Groebner Walk algorithm as described in
#   \cite{Collart:GroebnerWalk:97}.
#
# References:
#   Collart:GroebnerWalk:97
####################################################################
```

Since this source file header should contain the same information as for the online and manual documentation. It should be looked for a way to generate the header from one file that documents the function.

3.1.7 Standardizing Type Handling

Problem: Type checking within the parameter list of procedures is only used in the more recent **CASA** code (see Section 2.3 ("Types in **CASA**") for an overview over the **CASA** types).

Solution: Parameter type checking might be included for all top level procedures.

3.1.8 Introducing Revision Control

Problem: There is no revision control in the current version of **CASA**.

Solution: An RCS directory for every directory and every `private` subdirectory in `src` has to be created.

3.2 Goals Concerning the Documentation

In this section we outline the goals for improving the **CASA** system with respect to its documentation. We identified the following three goals:

1. unisourcification of documentation files,
2. creating documentation on "How to Write a **CASA** Package",
3. providing a new license file.

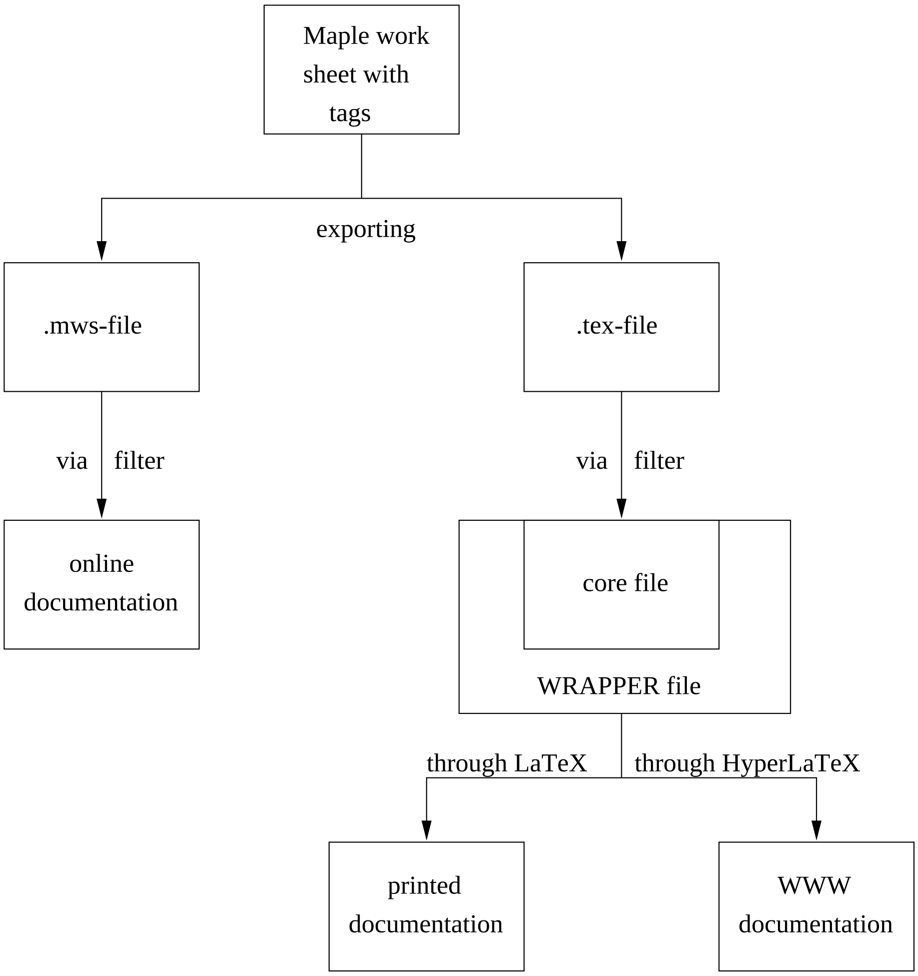
3.2.1 Unisourcification of Documentation Files

Problem: There exist three forms of **CASA** documentation: online documentation, web pages, and printed documentation. They are not created from one source. Hence they are not completely identical and their independence requires threefold work for every updating step concerning the documentation.

Solution: Starting from a **Maple** work sheet with tags, one can export it in two formats: as `.mws`- and as `.tex`-file.

The `.mws` file may then be processed to online documentation via a filter.

The `.tex` file may be processed to some "core file" by a filter. A wrapper file, which makes use of this core file may then be processed by LaTeX or Hyper-LaTeX in order to produce the printed documentation or the web documentation, respectively.



Work to be done: The filters (perl scripts) and the wrapper file mentioned in the above solution have to be written.

The wrapper file that is processed by LaTeX should be the same as the one processed by Hyper-LaTeX.

The **Maple** online manual produced by this solution would still be somehow independent from the web and printed documentation. Hence the goal is that these two formats of documentation have at least the same appearance.

3.2.2 Creating Documentation on "How to Write a CASA Package"

Problem: There exists no documentation on "How to Write a **CASA** Package".

Solution: Extending the current documentation by a section on "How to Write a **CASA** Package". This section should deal with conventions for packages, including topics as

- typing,
- file names, and
- error handling.

See also Subsection 3.1.4 ("Reorganization of Sub-packages") for conventions concerning sub-packages.

3.2.3 Providing a New License File

Problem: At the end of the `README` file that is distributed with **CASA** one finds the following copyright notice:

```
COPYRIGHT NOTICE
================

Copyright (C) 1990-1997 by
                Research Institute for Symbolic Computation
                Johannes Kepler University
                A-4040 Linz, Austria.

The system is freely distributed under the following condi-
tions: any research activity which uses CASA should cite the
authors and the system explicitly. The system can be freely
distributed to other users. New users are encouraged to no-
tify the CASA coordinator so that they can be included in a
user  list where they will be kept up to date about the pro-
gress of the system.

NOTE: This system is not a part of Maple share library.
```

In addition, the GNU General Public License is distributed with **CASA**. But **CASA** is not really distributed under the GPL conditions.

Solution : Extending or even replacing GNU's General Public License by an own **CASA** license.

Work to be done: A **CASA** license has to be formulated.

3.3 Long-Term Goals

In this section we outline some long-term goals for the **CASA** system. We identified the following four goals:

1. porting to **Maple** V.5,
2. providing a Windows NT installation,
3. creation of a test-suite,
4. improving the data representation for algebraic sets.

3.3.1 Porting to Maple V.5

Problem: The new **Maple** version **Maple** V.5 might have changes in some **Maple** procedures and data types that are used in **CASA**, and that affect the correctness of the current **CASA** code.

Solution: **CASA** has to be extensively tested with **Maple** V.5; bugs originating from the new **Maple** version have to be eliminated.

3.3.2 Providing Windows NT Installation

Problem: Currently, there exist only **CASA** installations under a Unix environment; experiences with installing **CASA** under Windows NT have not been made so far.

Solution: The installation procedure for **CASA** under Windows NT has to be explored and documented.

3.3.3 Creation of a Test-suite

Problem: Currently, there is no standardized test-suite that would allow to validate new installations or versions of **CASA** reliably.

Solution: A test-suite that satisfies the upper requirements has to be created. The extension of this test-suite has has to be done in cooperation with future **CASA** authors/contributors.

All existing executable **CASA** examples can be found in the directory `$demo`. They correspond to the non-executable examples in the online help (that can be viewed after executing ?`casa` in the **Maple** shell). The `.mws`-files that are used for the generation of the online documentation for **CASA** commands can be found in `$help/casa/mws`.

3.3.4 Improving the Data Representation for Algebraic Sets

Problem 1: Some attribute keys for the algebraic set type are not used in **CASA** (cf. Subsection 2.3.1).

Solution 1: Remove the keys from the files

`addattr`,
`mkImplAlgSet`,
`mkParaAlgSet`,
`mkPlacAlgSet`, and
`mkProjAlgSet`.

Problem 2: In **CASA** there exist four different representations for algebraic sets (cf Subsection 2.3.1). Some **CASA** procedures that have algebraic sets as input parameters assume them to be in a specific representation; otherwise a time-consuming conversion has to be performed.

Solution 2: The data structure for algebraic sets has to be designed in such a way that a once computed representation of an algebraic set is stored for future availability.

Bibliography

[1] R. Gebauer, M. Kalkbrener, B. Wall, and F. Winkler. CASA: A computer algebra package for constructive algebraic geometry. In S. M. Watt, editor, *Proceedings of ISSAC '91*, pages 403–410, Bonn, Germany, July 1991. ACM Press.

[2] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, and S. M. Vorkoetter. *Maple V Programming Guide*. Springer-Verlag, 1996. (release 4).

[3] M. Mňuk. *Algebraic and Geometric Approach to Parameterization of Rational Curves*. Phd thesis, Research Institute for Symbolic Computation, Linz, Austria, 1995.

[4] M. Mňuk and F. Winkler. CASA—a system for computer aided constructive algebraic geometry. In J. Calmet and C. Limongelli, editors, *Proceedings of the International Symposium DISCO'96*, number 1128 in LNCS, pages 297–307. Springer, 1996.

[5] Q.-N. Tran. A hybrid symbolic-numerical method for tracing surface-to-surface intersections. In A. H. M. Levelt, editor, *The Proceedings of ISSAC'95*, pages 51–58. ACM Press, 1995.

[6] Q.-N. Tran. On the symbolic-numerical methods for finding the roots of an arbitrary system of nonlinear algebraic equations. In *Proceedings of the First Asian Technology Conference in Mathematics, ATCM-95*, Singapore, 1995.

[7] Q.-N. Tran. *A Hybrid Symbolic-Numerical Approach in Computer Aided Geometric Design (CAGD) and Visualization*. Phd thesis, Research Institute for Symbolic Computation (RISC-Linz), University of Linz, Austria, 1996.

[8] Q.-N. Tran. Extending Newton's method for finding the roots of an arbitrary system of equations and its applications. *International Journal of Modeling and Simulation*, 17(4), 1997.

[9] Q.-N. Tran and F. Winkler. CASA reference manual (version 2.3). RISC Linz Report Series 97-33, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, Oct. 1997.

[10] Q.-N. Tran and F. Winkler. An overview of CASA—a system for computational algebra and constructive algebraic geometry. In T. V. Effelterre,

T. Recio, and F. Winkler, editors, *The Symbolic and Algebraic Computation (SAC) Newsletter*, volume 2. Stichting CAN, Computer Algebra Nederland, Universiteit van Amsterdam, the Netherlands, 1997.

[11] B. Wall. *Symbolic Computation with Algebraic Sets.* Phd thesis, Research Institute for Symbolic Computation (RISC-Linz), University of Linz, Austria, 1993.

[12] F. Winkler. Algebraic computation in geometry. *Special Issue of J. Math. in Comp. Simulation*, 42(4-6):529–537, 1996.