ABS
A Software for Abstraction-based Controller Synthesis

Gunther Reissig*, Alexander Weber, Elisei Macoveiciuc

User's Manual

Version 1 May 01, 2022

*Initiator and administrator of project, University of the Bundeswehr Munich, Dept. Aerospace Eng., Chair of Control Eng. (LRT-15), D-85577 Neubiberg (Munich), Germany, http://www.reiszig.de/gunther/

This work has been supported by the following sponsors: Deutsche Forschungsgemeinschaft (DFG) under grants no. RE 1249/3-1, RE 1249/3-2, and RE 1249/4-1; Munich Aerospace; and AdaCore.

Abstract

ABS is a software to solve synthesis, analysis and verification problems for infinite state dynamical systems. This document provides basic information on how to get, to compile and to use the software. For more detailed information, useful for programmers as well as for expert users, please refer to the companion document [1].

Publications

We hope you find this software or its underlying theory interesting, useful or even inspiring. If you do, we would appreciate it if you would acknowledge and cite our work:
The definitive publication presenting our software is [2]; a BibTeX entry can be found here:
www.reiszig.de/gunther/pubs/WeberMacoveiciucReissig22.html
The definitive publication presenting the underlying theory is [3]; a BibTeX entry can be found here:
www.reiszig.de/gunther/pubs/ReissigWeberRungger17.html

Acknowledgment

ABS is developed and maintained by the group of Gunther Reissig at the University of the Bundeswehr Munich. It is based on software created between 2013 and 2017 by Alexander Weber.

The following persons have substantially contributed to the development of ABS: Alexander Weber, Gunther Reissig, Hao Zhou, Elisei Macoveiciuc.

The following sponsors have supported the development, maintenance and provision of ABS: Deutsche Forschungsgemeinschaft (DFG) under grants no. RE 1249/3-1, RE 1249/3-2, and RE 1249/4-1; Munich Aerospace; and AdaCore.

Version and Revision Information

Version:	1.1
Revision:	2714
Date:	2022-05-01/16:28:38.386984Z
Branch Root:	https://subversion.unibw.de/lf1agure/ABS/tags/public/CurrentPublicVersion/trunk
Working Copy:	/home/reiszig/Software/ABS/tags/public/CurrentPublicVersion/trunk/
Revision of this document:	2710, 2022-05-01 16:16:05Z

Contents

1 Quick Start

ABS is a software to solve synthesis, analysis and verification problems for infinite state dynamical systems. Follow the steps below to quickly get a copy of ABS running on your computer, and to see it applied to an example.

Prerequisites. ABS is an implementation of the symbolic controller synthesis framework introduced in [3] and has been presented in [2]. Users are assumed to be familiar with both of the aforementioned documents.
ABS should be run on Linux, which is what we assume throughout this document. Some external software is required as well; we recommend to proceed with the following steps and follow the instructions printed out on the screen. If this does not work, please refer to [1, Section "System Prerequisites"] for more specific requirements.

Obtaining a copy of ABS. ABS is maintained using the version control system `svn` [4], and access is by user name and password. Create a working copy of ABS in a local directory of your choice, which may take some time:

```
> svn checkout --username ABSpublic  \
> https://subversion.unibw.de/lf1agure/ABS/tags/public/CurrentPublicVersion  \
> <full path to working copy of ABS>
```

If asked for it, you would use `IuseABS` as the password.

You have only read access to ABS; you are not allowed to commit to the repository hosting the software. However, if you are a member of the project administrator's group at the University of the Bundeswehr Munich, please refer to [1] for other ways to get (read and write) access.

License. ABS has been made publicly available under GNU General Public License. For terms and conditions, see the file `trunk/COPYING` in the working copy of ABS.

Solving a simple synthesis problem. Within the working copy, navigate to the directory `trunk/examples/SinglePole/` and run `make`. After some time and a continuous flow of messages, you are asked to choose problem and solution type: choose `0`. Then, you should eventually see a message similar to the following:

```
- Results:
  Control problem has been solved.
```

Navigate to the directory `trunk/examples/SinglePole/Results` to see two output files:

```
Controller.dat
Value_Function.dat
```

You may modify the control problem in the file `SinglePole.abcs` (open with a text editor) and start all over as described above. To delete all produced files, run

```
make clean
```

2 Introduction

ABS is an implementation of the symbolic controller synthesis framework introduced in [3]. The software applies to sampled-data control systems and to control problems with reach-avoid or invariance specification. It is based on the method [3, Th. VIII.4] for computing abstractions and the methods in [5, 6, 7] for controller synthesis. ABS is specifically described in [2]. This manual provides more details to the input, output and, more generally, the usage of the program.

2.1 Processable control problems

We briefly discuss the processable control problems below. See Fig. 1 for an illustration. The term control problem refers throughout to a pair of a control system ("plant") and a specification on that control system.

The plant is a sampled-data control system whose underlying continuous-time dynamics is of the form

$$\dot{x} \in f(x,u) + [\![-w,w]\!]\,, \tag{1}$$

where $f\colon \mathbb{R}^n \times U \to \mathbb{R}^n$ is continuous and locally Lipschitz continuous in the first argument, $U \subseteq \mathbb{R}^m$ is non-empty and $w \in \mathbb{R}^n_+$. $x\colon \mathbb{R}_+ \to \mathbb{R}^n$ and $u\colon \mathbb{R}_+ \to U$ denote the state and input signal, respectively.

ABS returns static state-feedback controllers $C\colon X \rightrightarrows U$ such that the obtain control law takes the form

$$u(t) \in C(P(x(t))), \tag{2}$$

where the map $P\colon X \rightrightarrows X$ given by $x \mapsto x + [\![-z,z]\!]$ for some constant $z \in \mathbb{R}^n_+$ models uncertainties in the measurement of x.

To discuss the processable specifications, let $F\colon \mathbb{R}^n \times U \rightrightarrows \mathbb{R}^n$ denote the right hand side of the discrete-time dynamics obtained from sampling (1) with a constant sampling time τ [3, Def. VIII.1]. In this way, ABS actually works with the discrete-time dynamics

$$x(t+1) \in F(x(t),u(t))$$

and time-discrete signals $x\colon \mathbb{Z}_+ \to \mathbb{R}^n$, $u\colon \mathbb{Z}_+ \to U$.

Reachability

A reach-avoid specification asks to steer each state signal (of the sampled system) starting in a set $A \subseteq \mathbb{R}^n$ into a target set $E \subseteq \mathbb{R}^n$ in finite time while avoiding an obstacle set $H \subseteq \mathbb{R}^n$. More formally, the condition is

$$x(t) \in A \implies \big(\, \exists_{T\in\mathbb{Z}_+}\, x(T) \in E \;\wedge\; \forall_{t\in[0;T]}\, x(t) \notin H \,\big). \tag{3}$$

and is represented by the triple (A,H,E) of subsets of the state space.

Invariance

An invariance specification asks to keep each state signal starting in a set A indefinitely in a target set E while avoiding an obstacle set H. To be precise, the condition is

$$x(0) \in A \implies \forall_{t\in\mathbb{Z}_+} x(t) \in E \setminus H \tag{4}$$

and is also represented by the triple (A,H,E) of subsets of the state space.

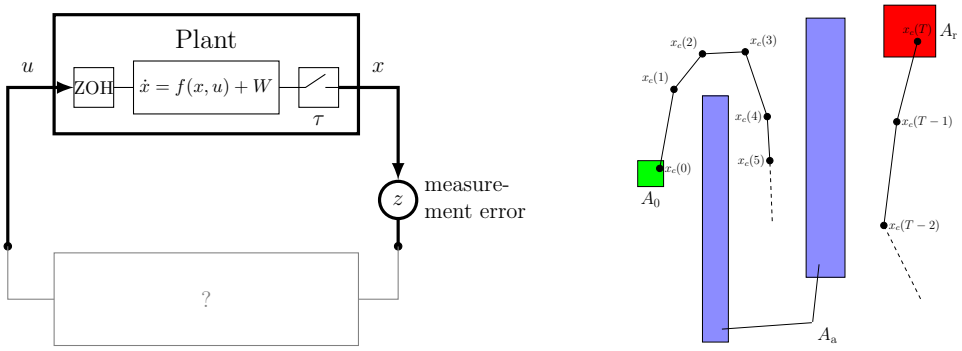


Figure 1: Illustration of the processable plant structure and a reach-avoid specification.

2.2 User input and program output

The specification of the control problem to solve is done by an ASCII file and using a special programming language (input language). All the problem-specific user input (including the control problem) is contained in that file, which we call thereafter *problem file*.

Every term written in a problem file is understood as a mathematical statement. That is, all the results returned from the software hold for the literal real numbers appearing in the problem file – not for floating-point approximations of the literals. For example, if the constant π (the ratio of the circumference of a circle to the diameter) appears in the definition of the function f in (1) then any given result is valid for π and not for some approximation of π. However, there is no guarantee that the software will be able to process or to solve the given control problem.

In the case of a successful controller synthesis the obtained controller is represented in an ASCII file. More precisely, the abstract controller and the quantizer according to [3, Th. VI.3] are specified in that file.

3 Specification of control problems

Subsequently, the formulation of control problems and of other required parameters is described.

3.1 Preparatory actions

In order to set up your own control problem the following actions must be taken:

1.) Run

```
make newproblem
```

in the root directory

2.) Enter a name for your new control problem

3.) Enter the path to an existing directory in your system. (Do not use the tilde ~ for pointing to your home directory.) In this directory, a subdirectory named as the name of your control problem will be automatically created. All the files related to your control problem (e.g. problem file, binary file) will be contained in that subdirectory. This subdirectory must not exist already

4.) Navigate to the previous subdirectory and open the file with file extension `abcs` using a text editor

5.) Specify your control problem and other parameters in the opened file using the programming language described in the next subsection

Example: Taking all but the last action above the command line may look as follows

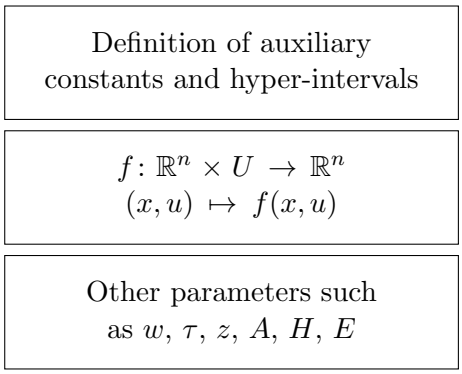
```
unibw@linux:~/1.2$ make newproblem
Please enter a name for your new control problem: Rocket
Please enter the path to an existing directory: ../ControlProblems
Specify your control problem in '../ControlProblems/Rocket/Rocket.abcs'.
unibw@linux:~/1.2$ emacs ../ControlProblems/Rocket/Rocket.abcs
```

3.2 Input language

We use in this section the notation of Section 2. We outline below the entities that the user has to specify:

Entity	Description	Type
n, m	State and input space dimension	Integer
z	Bound for measurement uncertainty, cf. (2)	Vector
$\bar{X}_1$	Operating range of controller	Hyper-interval
$\bar{U}$	Input space of right hand side of (1)	Hyper-interval
f	Unperturbed right hand side of (1)	Function
w	Bound for dynamic uncertainties, cf. (1)	Vector
τ	Sampling time	Real number
A, E, H	Initial, target, obstacle set, cf. (3), (4)	Union of hyper-intervals
d_1, d_2	State and input space discretization parameter	Vector
p, q	Integration orders for dynamics and growth bounds	Integer

Subsequently, we discuss the programming language with which the above entities are specified. At the same time, the restrictions on the entities above due to the programming language are explained. Thus, both the syntax and the semantics of the language are explained simultaneously. We actually discuss only the most relevant subset of the language. The full syntax is included in the appendix.




```
Real gamma in [0.0125,0.0126]; /* Friction param. */

f: (x,u) in (Real[2],[-2,2]) to y[2]
{
 y[0] = x[1];
 y[1] = -sin(x[0])-cos(x[0])*u-2*gamma*x[1];
}

SamplingTime : 0.3;
OperatingRange: ([0,2*Pi],[-2,2]);
ListOfPeriodicComponents : (0);
InitialSet : ([0,0],[0,0]);
TargetSet : ([Pi-.1,Pi+.1],[-.1,.1]);
InitialStateSpaceSubdivision : (100,100);
InitialInputSpaceSubdivision : (3);
IntegrationOrder : 5;
IntegrationOrderGrowthBound : 15;

/* This is a comment line */
```

Figure 2: problem file: Structure (l.h.s) and example (r.h.s.)

Every problem file has a structure as indicated in Fig. 2. In the first part, auxiliary constants and hyper-intervals are defined. The second part contains the specification of the function f in (1). In the last part, the remaining parameters are specified. A comment has to be enclosed inside the strings `/*` and `*/`, and can appear anywhere in the problem file. The file `examples/SinglePole/SinglePole.abcs` is listed in Fig. 2.

We use Backus-Naur Forms for specifying the grammar, where terminals are printed as

`terminal`

and nonterminals as

nonterminal

Keywords (or reserved words) are printed as

keyword

for better readability of the forms. The two components of productions are separated by the symbol ::= and the alternatives in the second component are separated by a vertical bar |. We refer the reader to [8] for the concept of context-free grammars. For example, we have the following productions in the grammar, which state, roughly speaking, that a digit is an Arabic numeral and an integer is a sequence of digits.

$$\begin{aligned} \textit{digit} &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \textit{literal\_unsigned\_integer} &::= \textit{digit} \mid \textit{literal\_unsigned\_integer}\ \textit{digit} \end{aligned}$$

3.2.1 Constants and intervals

A constant $c \in \mathbb{R}$ can be specified by

Real *identifier* = *expression* ;

where *expression* may represent an element of $\mathbb{R}$ or a term of finite compositions of the functions

+	Addition
−	Subtraction
×	Multiplication
÷	Division
^	Exponentiation
atan	Inverse tangent
cos	Cosine
cosh	Hyperbolic cosine
exp	Exponential
ln	Natural logarithm
sin	Sine
sinh	Hyperbolic sine
sqrt	Square root
tan	Tangent

See Appendix for the full specification of *expression* and *identifier*. For example, we may write

```
Real a = 2*Pi ;
```

for representing the circumference of the unit circle. Here, the keyword Pi represents π. An interval $[a, b] \subseteq \mathbb{R}$, $a \leq b$, can be specified using the production

literal_interval_expression ::= `[` *expression* `,` *expression* `]` `;`

where *expression* represents a and b respectively.

A constant $c \in \mathbb{R}$ can also be uncertain, in which case one can use

Real *identifier* in *literal_interval_expression* `;`

to express that c is bounded in an interval but its precise value is unknown.

It is also possible to define vectors $v \in \mathbb{R}^k$ and matrices $M \in \mathbb{R}^{k \times l}$, $k, l \in \mathbb{N}$. For example, the vector $v = (\sqrt{2}, 2^3) \in \mathbb{R}^2$ and the matrix in $M = \left(\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right) \in \mathbb{R}^{2 \times 2}$ are represented through the lines

```
Real v[2];
v[0] = sqrt(2);
v[1] = 2^3;
```

and

```
Real M[2][2];
M[0][0] = 1;
M[0][1] = 2;
M[1][0] = 3;
M[1][1] = 4;
```

respectively.

Important: Indices of components are zero-based.

3.2.2 Hyper-intervals

Hyper-intervals $[\![a, b]\!] = [a_1, b_1] \times \ldots \times [a_n, b_n]$, $a, b \in \mathbb{R}^n$, $a \leq b$, can be specified according

literal_hyperinterval_expression ::= `(` *interval_expression_list* `)` `;`

where *interval_expression_list* demands the specification of the intervals $[a_i, b_i]$, $i \in [1; n]$, according *literal_interval_expression* and written in succession separated by commas. $\mathbb{R}^n$ itself can also be considered a hyper-interval so that hyper-intervals in total can be specified according

hyperinterval_expression ::= *literal_hyperinterval_expression* | Real `[` *expression* `]` `;`

where *expression* shall represent n.

3.2.3 Continuous-time unperturbed dynamics

The syntax for specifying the right hand side in (1) is too complex to explain it using the actual productions. Therefore, we use, as it is the default in other programming books, an example. We might write

```
f: (x,u) in (Real[2],([-2,2],[0,1])) to y[2]
{
 y[0] = x[1] ;
 y[1] = - sin(x[0]) - cos(x[0])*u[0] - u[1]*x[1] ;
}
```

for representing the function $f\colon \mathbb{R}^2 \times U \to \mathbb{R}^2$ defined through

$$f(x,u) = (x_2, -\sin(x_1) - \cos(x_1) \cdot u_1 - u_2 x_2),$$

where $U = [-2,2] \times [0,1]$. We note that the identifiers `x` and `u` shall function in every problem file as the state and input variable. So, `x` and `u` are keywords of the input language. The dimension n shall appear to the right of the image variable (in the example code it is `y`) and the dimension m is implicitly specified through the hyper-interval that is associated to `u`.

Important: The input space U is also been defined through the previous code fragment. U shall be a compact hyper-interval.

Important: Indices of components are zero-based.

3.2.4 Bound on dynamic uncertainties

The bound on the dynamic uncertainties w is specified according

BoundOnDynamicUncertainties : *expression_tuple* ;

where *expression_tuple* represents an n-tuple of non-negative real numbers. For example,

```
BoundOnDynamicUncertainties : (0,1/2) ;
```

represents $w = (0, \frac{1}{2}) \in \mathbb{R}^2$.
This specification shall appear at most once in the problem file. Non-appearance means $w = 0$.

3.2.5 Sampling time

The sampling time τ is specified according

SamplingTime : *literal_unsigned_number* ;

where *literal_unsigned_number* shall be a positive real number.
This specification shall appear exactly once in the problem file.

3.2.6 Bound on measurement errors

The bound z on the measurement errors is specified according

BoundOnMeasurementErrors : *expression_tuple* ;

where *expression_tuple* represents an n-tuple of non-negative real numbers.
This specification shall appear at most once in the problem file. Non-appearance means $z = 0$.

3.2.7 Specifications

The triple (A, H, E) representing both a reachability problem and an invariance problem is specified as follows. Each of the sets A, H, E may be a finite union of compact hyper-intervals in $\mathbb{R}^n$, where every member of the union is specified according to the respective rules below. If k lines according to such a rule appear appropriately in the problem file then the set is a union of k (not necessarily distinct) hyper-intervals.

- A:

  InitialSet : *literal_hyperinterval_expression* ;

  This specification shall appear at least once.

- E:

  TargetSet : *literal_hyperinterval_expression* ;

  This specification shall appear at least once.

- H:

  ObstacleSet : *literal_hyperinterval_expression* ;

  Non-appearance of this specification means $H = \emptyset$.

3.2.8 Abstraction

- The (finite) set $\bar{X}_2$ is specified implicitly by specifying
  - a hyper-interval $[\![a, b]\!]$, $a, b \in \mathbb{R}^n$, $a < b$ according

    OperatingRange : *literal_hyperinterval_expression* ;

    This specification shall appear exactly once.
  - an integer vector $p \in \mathbb{N}^n$ according

    InitialStateSpaceSubdivision : *literal_unsigned_integer_tuple* ;

    This specification shall appear exactly once.

  The hyper-interval $[\![a, b]\!] = [a_1, b_1] \times \ldots \times [a_n, b_n]$ is the set $\cup_{x_2 \in \bar{X}_2} x_2$ and so $\bar{X}_2$ is implicitly defined by subdividing every edge $[a_i, b_i]$ of $[\![a, b]\!]$ into $d_{1,i}$ intervals of equal length. More concretely, $x_2 \in \bar{X}_2$ if and only if there exists $k \in \mathbb{Z}^n_+$ satisfying $k_i \in [0; d_{1,i} - 1]$ for every $i \in [1; n]$ and

  $$x_2 = a + v(k) + [\![0, \eta]\!] ,$$

  where $v(k)_i := k_i \eta_i$, $\eta_i = (b_i - a_i)/d_{1,i}$, $i \in [1; n]$.
  If there exists a subset $I \subseteq [1; n]$ so that f possesses the property

  $$f(x + (b_i - a_i)e_i) = f(x) \tag{5}$$

  for every $x \in \mathbb{R}^n$ and $i \in I$ then one may include a line according

  ListOfPeriodicComponents : *literal_unsigned_integer_tuple* ;

  to represent that property. In (5), e_i denotes the ith standard basis vector of $\mathbb{R}^n$.

  Note: The indices have to be specified zero-based.

- The (finite) set U_2 is specified implicitly by specifying an integer vector $p' \in \mathbb{N}^m$ according

  InitialInputSpaceSubdivision : *literal_unsigned_integer_tuple* ;

  The set U_2 is then obtained by subdividing the ith edge of U into $d_{2,i}$ intervals of equal length, $i \in [1; m]$. More concretely, $u_2 \in \bar{U}_2$ if and only if there exists $k \in \mathbb{Z}_+^m$ satisfying $k_i \in [0; d_{2,i} - 1]$ for every $i \in [1; m]$ and

  $$u_2 = a' + v(k) + \eta'/2,$$

  where $v(k)_i := k_i \eta'_i$, $\eta'_i = (b'_i - a'_i)/d_{2,i}$, $i \in [1; m]$, $U = [\![a', b']\!]$, $a' \leq b'$.

3.2.9 Order of integrations

The method [3, Th. VIII.4] involves the general solution of the unperturbed dynamics $\dot{x} = f(x, u)$. However, the general solution is usually not available as an explicit formula. Therefore, the implementation uses a Taylor polynomial as an approximation formula instead. In the current version, the degree of the polynomial (order of the approximation formula) has to be specified by the user. This is done according

IntegrationOrder : *literal_unsigned_integer* ;

where the integer shall be positive. Accordingly,

IntegrationOrderGrowthBound : *literal_unsigned_integer* ;

specifies the degree of the polynomial used to approximate the growth bound formula.

4 Controller synthesis

4.1 Starting the computation

Having specified the control problem and the other parameters the command

`make`

run in the problem directory will successively run

1.) the generation of problem-dependent **source code**

2.) the compilation of a binary file for the controller synthesis for this control **system**

3.) the **controller** synthesis

Usually, it is more convenient to launch the previous steps manually. (For example, when performing the first and second step on a laptop but the computationally most expensive last step on a remote workstation.) The following commands are provided for this purpose:

1.) `make sourcecode`

2.) `make system`

3.) `make controller`

Note: After `make controller` the user is asked to choose the actual algorithm to be applied. After that the actual computation starts. See Section 4.2.
Additionally, the command `make clean` will remove all files and directories that have been automatically generated by ABS. The problem file will not be removed.

4.2 User choices

In the current version, the user has the following choices:

0. On-the-fly solution to the reach avoid problem. Algorithms as in [6].
1. On-the-fly solution to the invariance problem. Algorithms as in [7].

   (!) Note that this option currently assumes that the set indicated as "OperatingRange" in the input file contains apriori enclosure of the forwards reachable set of the set indicated as "TargetSet". If this is not known, option 4. would assist the user to formulate the problem correctly following the steps:

   i. Construct an input file with the same problem data except for replacing numerical values to the right of "OperatingRange" with numerical values to the right of "TargetSet".
   ii. Run `make` and choose 'Compute apriori enclosure and quit'
   iii. Compare the set in the output (denoted here by A) with the set introduced as "OperatingRange" in the original input file (denoted here by B).
   iv. If $A \subseteq B$ then do not change anything in the original input file. Go to vi.
   v. If $B \setminus A \neq \emptyset$ then in the original input file replace numerical values to the right of "OperatingRange" (set B) with numerical values of the set $B \cup A$. Go to vi.
   vi. Run `make` , choose option 1.
2. Standard solution to the reach avoid problem. See [6, 3].
3. Standard solution to the invariance problem. See [7].
4. Compute apriori enclosure and quit. This auxiliary function output an apriori enclosure of the forward reachable set of the operating range. It may be useful to formulate invariance problems.

4.3 Output files

In the case of a successful controller synthesis the file `Controller.dat` will be created in the directory `Results`, where the latter is a subdirectory of the directory where the control problem has been specified. The file structure is as below.

line i	Content
$i = 1$	n
$i = 2$	m
$i = 3$	operating range
$i = 4$	U
$i = 5$	state space discretization
$i = 6$	input space discretization
$i = 7$	Information on encoding for line 9 on
$i = 8$	Information on encoding for line 9 on
$i \in [o; o + \lvert\bar{X}_2\rvert - 1]$, where $o = 9$	Index of the control symbol for the cell with index k, where $i = o + k$ is the corresponding line. A dash - indicates that the value function is either 0 or ∞ on the cell.

Appendix

For the understanding of the input language, the user should be familiar with context-free grammars and Backus-Naur forms [8] in the way those concepts are utilized for defining the syntax of programming languages. For example, the grammars for the C and Ada programming languages are included in [9] and [10], respectively. In the context of control theory, we remark that in [11, 12] the input language used therein is also defined using a context-free grammar.

It is important to note that the grammar is much larger than required for a problem file. The reason is that the grammar is also internally used for other purposes such as rounding error estimations. We want the grammar printed in this manual to coincide with the grammar in the implementation.

Grammar

Start symbol (Input file structure)

$$
\begin{aligned}
\textit{program} ::= \ & \textit{function\_definition} \\
| \ & \textit{function\_definition parameter\_list} \\
| \ & \textit{function\_definition ode\_definition} \\
| \ & \textit{statement\_list function\_definition} \\
| \ & \textit{statement\_list function\_definition parameter\_list} \\
| \ & \textit{statement\_list function\_definition ode\_definition}
\end{aligned}
$$

Statements

$$
\begin{aligned}
\textit{statement} ::= \ & \textit{declaration} \\
| \ & \textit{definition} \\
| \ & \textit{declaration\_and\_definition} \\
| \ & \textit{iteration\_statement} \\
\textit{declaration} ::= \ & \mathsf{Real}\ \textit{component\_expression}\ \mathsf{;} \\
| \ & \mathsf{Interval}\ \textit{component\_expression}\ \mathsf{;} \\
\textit{definition} ::= \ & \textit{component\_expression}\ \mathsf{=}\ \textit{expression}\ \mathsf{;} \\
| \ & \textit{component\_expression}\ \mathsf{=}\ \textit{literal\_interval\_expression}\ \mathsf{;} \\
| \ & \textit{component\_expression}\ \mathsf{in}\ \textit{interval\_expression}\ \mathsf{;} \\
\textit{declaration\_and\_definition} ::= \ & \mathsf{Real}\ \textit{identifier}\ \mathsf{=}\ \textit{expression}\ \mathsf{;} \\
| \ & \mathsf{Interval}\ \textit{identifier}\ \mathsf{=}\ \textit{literal\_interval\_expression}\ \mathsf{;} \\
| \ & \mathsf{Real}\ \textit{identifier}\ \mathsf{in}\ \textit{literal\_interval\_expression}\ \mathsf{;}
\end{aligned}
$$

$$
\begin{aligned}
\textit{compound\_statement} ::= \ & \{\ \textit{statement\_list}\ \} \\
\textit{iteration\_head} ::= \ & \mathsf{for}\ (\textit{identifier}\mathsf{=}\textit{expression}\,\mathsf{,}\,\textit{expression}\,\mathsf{,+}) \\
| \ & \mathsf{for}\ (\textit{identifier}\mathsf{=}\textit{expression}\,\mathsf{,}\,\textit{expression}\,\mathsf{,-}) \\
\textit{iteration\_statement} ::= \ & \textit{iteration\_head compound\_statement} \\
\textit{statement\_list} ::= \ & \textit{statement}
\end{aligned}
$$

| *statement_list statement*

Identifiers and literal nonnegative numbers

identifier ::= *nondigit*
| *identifier nondigit*
| *identifier digit*

literal_unsigned_integer ::= *digit* | *literal_unsigned_integer digit*

digit ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

nondigit ::= `a` | `b` | `c` | `d` | `e` | `f` | `g` | `h` | `i` | `j` | `k` | `l` | `m`
| `n` | `o` | `p` | `q` | `r` | `s` | `t` | `u` | `v` | `w` | `x` | `y` | `z`
| `A` | `B` | `C` | `D` | `E` | `F` | `G` | `H` | `I` | `J` | `K` | `L` | `M`
| `N` | `O` | `P` | `Q` | `R` | `S` | `T` | `U` | `V` | `W` | `X` | `Y` | `Z` | `_`

literal_unsigned_decimal ::= *literal_unsigned_integer* `.` *literal_unsigned_integer*
| `.` *literal_unsigned_integer*
| *literal_unsigned_integer* `.`
| *literal_unsigned_integer* `.` *literal_unsigned_integer* `E` `+` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` *literal_unsigned_integer* `E` `-` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` `E` `+` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` `E` `-` *literal_unsigned_integer*
| `.` *literal_unsigned_integer* `E` `+` *literal_unsigned_integer*
| `.` *literal_unsigned_integer* `E` `-` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` *literal_unsigned_integer* `e` `+` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` *literal_unsigned_integer* `e` `-` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` `e` `+` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` `e` `-` *literal_unsigned_integer*
| `.` *literal_unsigned_integer* `e` `+` *literal_unsigned_integer*
| `.` *literal_unsigned_integer* `e` `-` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` *literal_unsigned_integer* `E` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` `E` *literal_unsigned_integer*
| `.` *literal_unsigned_integer* `E` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` *literal_unsigned_integer* `e` *literal_unsigned_integer*
| *literal_unsigned_integer* `.` `e` *literal_unsigned_integer*
| `.` *literal_unsigned_integer* `e` *literal_unsigned_integer*

Expressions

component_expression ::= *identifier*
| *identifier* `[` *expression* `]`
| *identifier* `[` *expression* `]` `[` *expression* `]`

expression ::= *multiplicative_expression*
| *expression* + *multiplicative_expression*
| *expression* - *multiplicative_expression*

multiplicative_expression ::= *signed_expression*
| *multiplicative_expression* * *signed_expression*
| *multiplicative_expression* / *signed_expression*

signed_expression ::= *exponential_expression*
| - *exponential_expression*
| + *exponential_expression*

exponential_expression ::= *primary_expression*
| *primary_expression* ^ *signed_expression*

primary_expression ::= *component_expression*
| *literal_unsigned_number*
| *elementary_function_expression*
| (*expression*)

elementary_function_expression ::= *name_of_elementary_function*(*expression*)

name_of_elementary_function ::= atan | cos | cosh | exp | ln | sin | sinh | sqrt | tan

literal_unsigned_number ::= *literal_unsigned_integer*
| *literal_unsigned_decimal*
| Pi

Identifier and component expression lists

identifier_list ::= *identifier*
| *identifier_list* , *identifier*

identifier_tuple ::= (*identifier_list*)

component_expression_list ::= *component_expression*
| *component_expression_list* , *component_expression*

component_expression_tuple ::= (*component_expression_list*)

Intervals

literal_interval_expression ::= [*expression* , *expression*]

interval_expression ::= *component_expression*
| *literal_interval_expression*
| Real

interval_expression_list ::= *interval_expression*
| *interval_expression_list* , *interval_expression*

Hyper-intervals

literal_hyperinterval_expression ::= (*interval_expression_list*)
hyperinterval_expression ::= *literal_hyperinterval_expression*
| Real [*expression*]
interval_hyperinterval_expression_list ::= *interval_expression*
| *hyperinterval_expression*
| *interval_hyperinterval_expression_list* , *interval_expression*
| *interval_hyperinterval_expression_list* , *hyperinterval_expression*
interval_hyperinterval_expression_tuple ::= (*interval_hyperinterval_expression_list*)

Explicitly defined function

function_definition ::= *function_declaration function_head compound_statement*
function_declaration ::= *identifier* :
function_image ::= *component_expression_tuple*
| *component_expression*
function_head ::= *identifier* in *interval_expression* to *function_image*
| *identifier* in *hyperinterval_expression* to *function_image*
| *identifier_tuple* in *interval_hyperinterval_expression_tuple* to *function_image*

Ordinary differential equation

ode_definition ::= *function_declaration function_head ode_statement*
ode_statement ::= { *ode_equation* ; *initialvalue_list* }
| { *initialvalue_list ode_equation* ; }
ode_equation ::= diff (*identifier* , *identifier*) = *identifier identifier_tuple*
| diff (*identifier_tuple* , *identifier*) = *identifier identifier_tuple*
initialvalue ::= initialvalue (*component_expression* , *expression*) ;
initialvalue_list ::= *initialvalue*
| *initialvalue_list initialvalue*

Parameters

literal_unsigned_integer_list ::= *literal_unsigned_integer*
| *literal_unsigned_integer_list* , *literal_unsigned_integer*
literal_unsigned_integer_tuple ::= (*literal_unsigned_integer_list*)

expression_list ::= *expression*
| *expression_list* , *expression*

expression_tuple ::= (*expression_list*)

parameter_list ::= *parameter*
| *parameter_list parameter*

parameter ::= ListOfPeriodicComponents : *literal_unsigned_integer_tuple* ;
| InitialStateSpaceSubdivision : *literal_unsigned_integer_tuple* ;
| InitialInputSpaceSubdivision : *literal_unsigned_integer_tuple* ;
| SamplingTime : *literal_unsigned_number* ;
| InitialSet : *literal_hyperinterval_expression* ;
| TargetSet : *literal_hyperinterval_expression* ;
| ObstacleSet : *literal_hyperinterval_expression* ;
| OperatingRange : *literal_hyperinterval_expression* ;
| BoundOnDynamicUncertainties : *expression_tuple* ;
| BoundOnMeasurementErrors : *expression_tuple* ;
| IntegrationOrder : *literal_unsigned_integer* ;
| IntegrationOrderGrowthBound : *literal_unsigned_integer* ;

References

[1] G. Reissig, A. Weber, and H. Zhou, *ABS – A Software for Abstraction-based Controller Synthesis*, Bundeswehr University Munich, Programmer's Manual.

[2] A. Weber, E. Macoveiciuc, and G. Reissig, "ABS: A formally correct software tool for space-efficient symbolic synthesis," in *Proc. 25th ACM Intl. Conf. on Hybrid Systems: Computation and Control (HSCC), Milan, Italy, May 4-6, 2022*, 2022. doi:10.1145/3501710.3519519

[3] G. Reissig, A. Weber, and M. Rungger, "Feedback refinement relations for the synthesis of symbolic controllers," *IEEE Trans. Automat. Control*, vol. 62, no. 4, pp. 1781–1796, Apr. 2017. doi:10.1109/TAC.2016.2593947

[4] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*, for subversion 1.7 ed., 2011, http://svnbook.red-bean.com/.

[5] G. Reissig and M. Rungger, "Symbolic optimal control," *IEEE Trans. Automat. Control*, vol. 64, no. 6, pp. 2224–2239, Jun. 2019. doi:10.1109/TAC.2018.2863178

[6] E. Macoveiciuc and G. Reissig, "Memory efficient symbolic solution of quantitative reach-avoid problems," in *Proc. American Control Conference (ACC), Philadelphia, U.S.A., 10-12 Jul. 2019*, 2019, pp. 1671–1677. doi:10.23919/ACC.2019.8814850

[7] E. Macoveiciuc and G. Reissig, "Guaranteed memory reduction in synthesis of correct-by-design invariance controllers," in *Proc. 21st IFAC World Congress, Berlin, Germany, Jul. 12-17, 2020*, ser. IFAC-PapersOnLine, vol. 53, no. 2, 2020, pp. 5561–5566. doi:10.1016/j.ifacol.2020.12.1567

[8] S. Ginsburg, *The mathematical theory of context-free languages.* McGraw-Hill Book Co., New York-London-Sydney, 1966.

[9] B. W. Kernighan and D. M. Ritchie, *The C programming language*, 2nd ed., ser. Prentice-Hall software series. Prentice Hall, 1 Apr. 1988.

[10] J. Barnes, *Programming in Ada 2012.* New York, NY, USA: Cambridge University Press, 2014.

[11] S. Mouelhi, A. Girard, and G. Gössler, "CoSyMA: A tool for controller synthesis using multi-scale abstractions," in *Proc. 16th Intl. Conf. Hybrid Systems: Computation and Control (HSCC), Philadelphia, PA, U.S.A., Apr. 8-11, 2013*, 2013, pp. 83–88.

[12] S. Mouelhi, A. Girard, and G. Gössler, *CoSyMA: A Tool for Controller Synthesis Using Multi-scale Abstractions*, INRIA, Oct. 2012, Research Report no 8108.