Centre for Discrete Mathematics and Computing

School of Computer Science & Electrical Engineering
and Department of Mathematics,
The University of Queensland, QLD 4072

Technical Report #14

Title: ACE for Amateurs (Version 3.001)

Author: Colin Ramsay

Date: April 4, 2001
Version: draft (incomplete)

Contents

List of figures

List of tables

Chapter 1

Introduction

ACE is designed to work with partial tables, as well as complete tables exhibiting a finite index. TBA: Intended user groups ...

ACE is divided into three 'levels'. The actual enumerator, called the "core enumerator", is ACE Level 0, while the standard driver for the enumerator, the "core wrapper", is ACE Level 1. A stand-alone 'example' application, called the "interactive interface", is ACE Level 2. To assist those interested in the actual source code, the function and variable names are prepended with `AL0_`, `AL1_` & `AL2_` respectively. ACE also includes the "proof table" package (PT for short), which can be compiled into the executable if required. The proof table cuts across the level structure, and can only be used as part of the interactive interface. Function and variable names of the PT package are prepended with `PT_`. TBA: this package ...

TBA: version history, 3.000 vs 3.001, ... TBA: default build ...

1.1 Administrivia

It is assumed that ACE is run on a Unix-box of some description. TBA: how to compile ...

In order not to clutter-up the body of the text with examples, the bulk of these are gathered into a separate appendix. These examples illustrate many of the features of ACE, and can also serve as a source of interesting enumerations. Some are referred to in the text, but they can all be read independently. ACE script input generating these examples is available in the `ex***.in` files, as part of the documentation.

1.2 Code

You will note in the source code various sections preceded by a warning comment containing the `DTT` acronym. This stands for "debug/test/trace", and denotes code that was added temporarily for one reason or another. None of this code should be active; i.e., it should all be commented out. It does *not* form part of the ACE distribution. Of course, gurus will find this code intriguing, and will probably want to uncomment it to see what happens!

TBA: The source code is heavily commented, and is considered to be part of the documentation. Conceptually, coset enumeration is easy, but there are tricky details and subtle performance issues – you need to read the source code, to experiment, and to think to appreciate these.

Chapter 2

Background

2.1 Terminology

Although ACE can accept either letters or numbers for group generators, we generally use letters, since these are much easier to understand. (Unless you need more than 26 generators, or are using some form of automatically generated presentation, you should adopt the same convention.) Lower-case letters denote generators, with inverses being denoted by either upper-case letters or negative superscripts; e.g., $ABab$ and $a^{-1}b^{-1}ab$ are equivalent. We use 1 to denote the identity element and/or the subgroup (i.e., coset #1).

...scanning, applying, closing. ...dead coset(s), compact(ion).

2.2 Notation

For a subgroup H of a group G, we represent by G/H the set of *right* cosets of H in G (*not* the quotient group of G by H which requires that H be normal in G), i.e.

$$G/H = \{Hx \mid x \in G\}.$$

Two cosets $Hx, Hy \in G/H$ are equal, i.e. coincident, if and only if $xy^{-1} \in H$. Also, any two cosets of G/H are either coincident or disjoint. The cardinality of G/H is the number of distinct cosets in G/H, and is equal to the index $|G : H|$ of H in G; if G is finite then $|G : H| = |G|/|H|$.

Some standard groups that arise in our examples are:

S_n, the full *symmetric group* on n letters;
A_n, the full *alternating group* on n letters; and
C_n, the *cyclic group* of order n.

A group G will often be defined via a *presentation* of the form

$$\langle \textit{generators} \mid \textit{relators} \rangle.$$

In this case, the elements of G are words in the *generators* and the *relators* are a list of words that are equivalent to the empty word (i.e. identity element) in G. Actually, amongst the *relators* we will also allow *relations*, which are equations of the form $w_1 = w_2$ (equivalent to the relator $w_1w_2^{-1}$), where w_1, w_2 are words in the generators of G.

Table 2.1: The coset table for S_4/S_3

	Generators		
coset	b_1	b_2	b_3
O_4 (G_3)	O_4	O_4	O_3
O_3 (G_3b_3)	O_3	O_2	O_4
O_2 $(G_3b_3b_2)$	O_1	O_3	O_2
O_1 $(G_3b_3b_2b_1)$	O_2	O_1	O_1

2.3 History

The concept of a subgroup, and its cosets, has been known since the beginnings of group theory. One of the earliest (practical?) uses of cosets seems to have been by Moore [12], who gives presentations for S_n & A_n and proves them correct by, in effect, counting the n cosets of S_n/S_{n-1} & A_n/A_{n-1}. Dickson [5, §264] presents a more accessible account, and explicitly notes that "these sets form a rectangular table". To illustrate this, we paraphrase Dickson's proof for the case S_4.

Let G_4 be the abstract group

$$\langle b_1, b_2, b_3 \mid b_1^2, b_2^2, b_3^2, b_1b_3 = b_3b_1, b_1b_2b_1 = b_2b_1b_2, b_2b_3b_2 = b_3b_2b_3\rangle.$$

Now S_4 is generated by the transpositions $s_1 = (12)$, $s_2 = (23)$ & $s_3 = (34)$. Putting $s_i = b_i$, $1 \leq i \leq 3$, we see that these transpositions satisfy the defining relations of G_4. So S_4 is a quotient group of G_4, and $|G_4| \geq |S_4| = 4! = 24$.

That $|G_4| \leq 24$, and so $G_4 \cong S_4$, is proved by induction. Let G_3 be the subgroup of G_4 generated by b_1 & b_2. (The actual induction is on the b_i. For our purposes, we'll simply assume that $|G_3| \leq 6$.) Now consider the cosets $O_4 = G_3$, $O_3 = G_3b_3$, $O_2 = G_3b_3b_2$ & $O_1 = G_3b_3b_2b_1$. We'll show that these four cosets are merely permuted by the b_i, so that the index $|G_4 : G_3| \leq 4$; hence $|G_4| \leq 24$, as required.

Obviously, $O_4b_3 = O_3$, $O_3b_2 = O_2$ & $O_2b_1 = O_1$. Since the b_i are involutions, then $O_3b_3 = G_3b_3b_3 = G_3 = O_4$. Similarly, $O_2b_2 = O_3$ & $O_1b_1 = O_2$. Since b_1 & b_2 generate G_3, then $O_4b_1 = O_4b_2 = O_4$. Now, since b_1 & b_3 commute, then $O_3b_1 = G_3b_3b_1 = G_3b_1b_3 = O_4b_1b_3 = O_4b_3 = O_3$. Now consider $O_1b_3 = G_3b_3b_2b_1b_3 = G_3b_3b_2b_3b_1$. Since $b_2b_3b_2 = b_3b_2b_3$, then this can be written as $G_3b_2b_3b_2b_1 = O_4b_2b_3b_2b_1 = O_4b_3b_2b_1 = O_1$. In a similar manner, $O_1b_2 = O_1$ & $O_2b_3 = O_2$. Our coset table (see Table 2.1) is now complete, so $|G_4 : G_3| \leq 4$. Note that each b_i gives rise to the transposition (O_iO_{i+1}), and leaves the other cosets fixed.

The construction of a coset table was systematised and popularised by Todd & Coxeter [16]. The first computer implementation was that of Haselgrove in 1953. This, along with other early implementations, is described by Leech [9]. Detailed accounts of the techniques used in coset enumeration can be found in [3, 6, 10, 13, 15]. Formal proofs of the correctness of various strategies for coset enumeration are given in [11, 13, 15].

Chapter 3

ACE Level 2: an interactive interface

Level 2 of ACE is a complete, standalone application for generating and manipulating coset tables. It can be used interactively, or can take its input from a script file. It is reasonably robust and comprehensive, but no attempt has been made to make it 'industrial strength' or to give it any of the features of, say, Magma [2] or GAP [14]. Most of its features have been added in response to user requests, and it is assumed that the user is 'competent'. One of the primary goals in developing ACE was to demonstrate how to correctly use ACE Levels 0 & 1; some care is taken to ensure that the user cannot generate 'invalid' tables.

A complete description of all the Level 2 commands is given in Appendix B.

3.1 Enumeration mode & style

The core enumerator takes two arguments, which select the enumeration mode and style. The mode determines whether or not we retain any existing table information. Initially, we start with an empty table and use the begin mode (the `beg` command). This can be followed by a series of continue and/or redo modes (the `cont` & `redo` commands) which build on or modify the table generated by the begin mode. So it is possible to do an enumeration in stages, altering the parameters at each stage. Various interlocks are present to prevent a combination of choices which (potentially) leads to an invalid table.

The enumeration style is the balance between C-style definitions (i.e., coset table based, Felsch style) and R-style definitions (i.e., relator based, HLT style), and is controlled by the `ct` & `rt` parameters. The absolute values of these parameters sets the number of definitions (C-style) or coset applications (R-style) per pass through the enumerator's main loop. The sign of these parameters sets the style, and the possible combinations are given in Table 3.1

In R style all the definitions are made via relator scans; i.e., this is HLT mode. In C style all the definitions are made in the next empty table slot and are tested in all essentially different positions in the relators; i.e., this is Felsch mode. In R/C style we run in R style until an overflow, perform a lookahead on the entire table, and then switch to CR style. Defaulted R/C style is the default style, and here we use R/C style with `ct:1000` and `rt` set to approximately 2000 divided by the total

Table 3.1: The styles

`Rt` value	`Ct` value	style name
<0	<0	R/C
<0	0	R*
<0	>0	Cr
0	<0	C
0	0	R/C (defaulted)
0	>0	C
>0	<0	Rc
>0	0	R
>0	>0	CR

length of the relators, in an attempt to balance R & C definitions when we switch to CR style. Rc & Cr styles are like R & C styles, except that a single C or R style pass (respectively) is done after the initial R or C style pass. R* style makes definitions the same as R style, but tests all definitions as for C style. In CR style alternate passes of C style and R style are performed, with all definitions tested. The `Ct` < 0 C style is reserved for future use, and should not be used.

3.2 Predefined strategies

The versatility of ACE means that it can be difficult to select appropriate parameters when presented with a new enumeration. The problem is compounded by the fact that no generally applicable rules exist to predict, given a presentation, which parameter settings are 'good'. To help overcome this problem, ACE contains various commands which select particular enumeration strategies. One or other of these strategies may work and, if not, the results may indicate how the parameters can be varied to obtain a successful enumeration. The thirteen standard strategies are listed in Table 3.2.

Note that we explicitly (re)set all of the listed enumerator parameters in all of the predefined strategies, even although some of them have no effect. For example, the `fi` value is irrelevant in HLT mode. The idea behind this is that, if you later change some parameters individually, then the enumeration retains the 'flavour' of the last selected predefined strategy. Note also that other parameters which may affect an enumeration are left untouched by setting one of the predefined strategies; for example, the values of `max` & `asis`. These parameters have an effect which is independent of the selected strategy.

Note that, apart from the `fel:0` & `sims:9` strategies, all of the strategies are distinct, although some are very similar. Further details of each strategy are contained in their entry in Appendix B.

Table 3.2: The predefined strategies

strategy	path	row	mend	no	look	com	ct	rt	fi	pmod	psiz	dmod	dsiz
	parameter												
def	n	y	n	-1	n	10	0	0	0	3	256	4	1000
easy	n	y	n	0	n	100	0	1000	1	0	256	0	1000
fel:0	n	n	n	0	n	10	1000	0	1	0	256	4	1000
fel:1	n	n	n	-1	n	10	1000	0	0	3	256	4	1000
hard	n	y	n	-1	n	10	1000	1	0	3	256	4	1000
hlt	n	y	n	0	1	10	0	1000	1	0	256	0	1000
pure c	n	n	n	0	n	100	1000	0	1	0	256	4	1000
pure r	n	n	n	0	n	100	0	1000	1	0	256	0	1000
sims:1	n	y	n	0	n	10	0	1000	1	0	256	0	1000
sims:3	n	y	n	0	n	10	0	-1000	1	0	256	4	1000
sims:5	n	y	y	0	n	10	0	1000	1	0	256	0	1000
sims:7	n	y	y	0	n	10	0	-1000	1	0	256	4	1000
sims:9	n	n	n	0	n	10	1000	0	1	0	256	4	1000

Chapter 4

ACE Level 1: a core wrapper

ACE Level 0 is a complete, efficient coset enumerator. However, it is ‘naked’, in the sense that it expects all its data structures to be correctly setup and it assumes that it is ‘sensibly’ driven. ACE Level 1 is a simple wrapper for Level 0 which processes the presentation and the parameters, and sets up the appropriate data structures. It contains some utility routines to help drive ACE, and it prevents some of the more obvious errors. Although it has to be used with care, the wrapper is a great deal easier to drive than the core enumerator, and is its recommended interface.

Chapter 5

ACE Level 0: the core enumerator

TBA: …

Appendix A

Examples

In this appendix we give some examples of ACE runs. A stand-alone discussion of some of the features of these runs is included, although parts of these runs are mentioned in the body of the text, as illustrations of specific features of ACE's behaviour. The `ex***.in` files supplied as part of this documentation can be used to run these examples, although an example may be presented as if it were generated interactively, and the output may be edited for reasons of space or perspicuity. There may be minor variations in the exact format of the output, since ACE is continually being 'improved'. Unless otherwise noted, all parameters are defaulted and the default build of ACE was used. In multipart runs, note that parameters from an earlier part may carry across to a later one. Note that some of the examples may require a machine with a large amount of memory.

A.1 Getting started

This example uses input file `ex000.in`, and illustrates the basics of ACE. Note how the input is generally insensitive to command synonyms, capitalisation, white space, and : & ; characters. When ACE starts up, it prints out its version, the date & time, and the name of the host on which it's running. If we attempt to do an enumeration immediately we get an error, since the lack of generators means we can't build the (empty) coset table.

```
ACE 3.001        Wed Apr  4 22:35:40 2001
=========================================
Host information:
  name = mango
end;
** ERROR (continuing with next line)
   can't start (no generators?)
```

After defining two generators, we can do an enumeration. The default state is not to echo the presentation or print any messages; only the result line is printed. The group is free, since there are no relators, and the subgroup is trivial. So the enumeration overflows.

```
gr:ab;          # A stupid comment
Begin
OVERFLOW (a=249998 r=83333 h=83333 n=249999; l=337 c=0.15; m=249998 t=249998)
```

The sr commands dumps out the presentation and the parameters for the run. All of these are currently defaulted, apart from those dependent on there being two (non-involutionary) generators.

```
sr:1;
  #--- ACE 3.001: Run Parameters ---
Group Name: G;
Group Generators: ab;
Group Relators: ;
Subgroup Name: H;
Subgroup Generators: ;
Wo:1000000; Max:249998; Mess:0; Ti:-1; Ho:-1; Loop:0;
As:0; Path:0; Row:1; Mend:0; No:0; Look:0; Com:10;
C:0; R:0; Fi:7; PMod:3; PSiz:256; DMod:4; DSiz:1000;
  #---------------------------------
```

With `sr:2` only the `Group Name` line is printed. Similarly, `sr:3`, `sr:4` and `sr:5` print the `Group Relators`, `Subgroup Name` and `Subgroup Generators` lines, respectively.

```
sr:2;
Group Name: G;
```

Next we print out the first part of the table. Note that, as there are no relators, the table has separate columns for generator inverses. So the default workspace of 1000000 words allows a table of $249998 = 1000000/4-2$ cosets. As row filling is on by default, the table is simply filled with cosets in order. Note that a compaction phase is done before printing the table, but that this does nothing here (the lower-case `co` tag), since there are no dead cosets. The coset representatives are simply all possible freely reduced words, in length plus lexicographic order.

```
pr:-1,12;
co: a=249998 r=83333 h=83333 n=249999; c=+0.00
 coset |      a      A      b      B   order   rep've
-------+---------------------------------------------
     1 |      2      3      4      5
     2 |      6      1      7      8       0   a
     3 |      1      9     10     11       0   A
     4 |     12     13     14      1       0   b
     5 |     15     16      1     17       0   B
     6 |     18      2     19     20       0   aa
     7 |     21     22     23      2       0   ab
     8 |     24     25      2     26       0   aB
     9 |      3     27     28     29       0   AA
    10 |     30     31     32      3       0   Ab
    11 |     33     34      3     35       0   AB
    12 |     36      4     37     38       0   ba
```

We now set things up to do the alternating group on five letters, of order 60. We turn messaging on, but set the interval high enough so that there will be no progress messages.

```
Enum: A_5;
rel: a^2, b^3, ababababab;
subgr: trivial;
mess: 1000;    start;
```

The presentation and the parameters are echoed, the enumeration is performed, and then the results of the run are printed. Note that the exponent of the `ababababab` word has been correctly deduced, and that `a` is treated as an involution. So the table has only three columns now. Definitions are HLT-style, and a total of 76 cosets (incl. the subgroup) are defined.

```
  #--- ACE 3.001: Run Parameters ---
Group Name: A_5;
Group Generators: ab;
Group Relators: (a)^2, (b)^3, (ab)^5;
Subgroup Name: trivial;
Subgroup Generators: ;
Wo:1000000; Max:333331; Mess:1000; Ti:-1; Ho:-1; Loop:0;
As:0; Path:0; Row:1; Mend:0; No:3; Look:0; Com:10;
C:0; R:0; Fi:6; PMod:3; PSiz:256; DMod:4; DSiz:1000;
  #---------------------------------
INDEX = 60 (a=60 r=77 h=1 n=77; l=3 c=0.01; m=66 t=76)
```

We now use a non-trivial subgroup, and monitor all the actions of the enumerator.

```
Subgroup Name: C_5 ;
gen:ab;
Monit :1
END;
  #--- ACE 3.001: Run Parameters ---
Group Name: A_5;
Group Generators: ab;
Group Relators: (a)^2, (b)^3, (ab)^5;
Subgroup Name: C_5;
Subgroup Generators: ab;
Wo:1000000; Max:333331; Mess:1; Ti:-1; Ho:-1; Loop:0;
As:0; Path:0; Row:1; Mend:0; No:3; Look:0; Com:10;
C:0; R:0; Fi:6; PMod:3; PSiz:256; DMod:4; DSiz:1000;
  #---------------------------------
AD: a=2 r=1 h=1 n=3; l=1 c=+0.00; m=2 t=2
SG: a=2 r=1 h=1 n=3; l=1 c=+0.00; m=2 t=2
RD: a=3 r=1 h=1 n=4; l=2 c=+0.00; m=3 t=3
RD: a=4 r=2 h=1 n=5; l=2 c=+0.00; m=4 t=4
RD: a=5 r=2 h=1 n=6; l=2 c=+0.00; m=5 t=5
RD: a=6 r=2 h=1 n=7; l=2 c=+0.00; m=6 t=6
RD: a=7 r=2 h=1 n=8; l=2 c=+0.00; m=7 t=7
RD: a=8 r=2 h=1 n=9; l=2 c=+0.00; m=8 t=8
RD: a=9 r=2 h=1 n=10; l=2 c=+0.00; m=9 t=9
CC: a=8 r=2 h=1 n=10; l=2 c=+0.00; d=0
RD: a=9 r=5 h=1 n=11; l=2 c=+0.00; m=9 t=10
RD: a=10 r=5 h=1 n=12; l=2 c=+0.00; m=10 t=11
RD: a=11 r=5 h=1 n=13; l=2 c=+0.00; m=11 t=12
RD: a=12 r=5 h=1 n=14; l=2 c=+0.00; m=12 t=13
RD: a=13 r=5 h=1 n=15; l=2 c=+0.00; m=13 t=14
RD: a=14 r=5 h=1 n=16; l=2 c=+0.00; m=14 t=15
CC: a=13 r=6 h=1 n=16; l=2 c=+0.00; d=0
CC: a=12 r=6 h=1 n=16; l=2 c=+0.00; d=0
INDEX = 12 (a=12 r=16 h=1 n=16; l=3 c=0.00; m=14 t=15)
```

We now dump out the statistics accumulated during the run. The run had `a=12` & `t=15`, so there must have been three coincidences (`qcoinc=3`). Of these, two were primary coincidences (`rdcoinc=2`). Since `t=15`, there must have been fourteen coset definitions; one was during the application of coset #1 (i.e., the subgroup) to the subgroup generator (`apdefn=1`), and the remainder during the application of the cosets to the relators (`rddefn=13`).

```
STATistics;
  #- ACE 3.001: Level 0 Statistics -
cdcoinc=0 rdcoinc=2 apcoinc=0 rlcoinc=0 clcoinc=0
  xcoinc=2 xcols12=4 qcoinc=3
  xsave12=0 s12dup=0 s12new=0
  xcrep=6 crepred=0 crepwrk=0 xcomp=0 compwrk=0
xsaved=0 sdmax=0 sdoflow=0
xapply=1 apdedn=1 apdefn=1
rldedn=0 cldedn=0
xrdefn=1 rddedn=5 rddefn=13 rdfill=0
xcdefn=0 cddproc=0 cdddedn=0 cddedn=0
  cdgap=0 cdidefn=0 cdidedn=0 cdpdl=0 cdpof=0
  cdpdead=0 cdpdefn=0 cddefn=0
  #---------------------------------
```

Note how the pre-printout compaction phase now does some work (the upper-case `CO` tag), since there were coincidences, and hence dead cosets. Note how `b/B` have been used as the first two columns, since these must be occupied by a generator/inverse pair or a pair of involutions. The `a` column is also the `A` column, as `a` is an involution. (Using `asis` and inputting the `a^2` relator as `aa`, however, stops ACE from treating `a` as an involution and the columns are not reordered. We will see this later.)

```
  print  TABLE : -1, 12 ;
CO: a=12 r=13 h=1 n=13; c=+0.00
 coset |      b      B      a   order   rep've
-------+-------------------------------------
     1 |      3      2      2
     2 |      1      3      1       3   B
     3 |      2      1      4       3   b
     4 |      8      5      3       5   ba
     5 |      4      8      6       2   baB
     6 |      9      7      5       5   baBa
     7 |      6      9      8       3   baBaB
     8 |      5      4      7       5   bab
     9 |      7      6     10       5   baBab
    10 |     12     11      9       3   baBaba
    11 |     10     12     12       2   baBabaB
    12 |     11     10     11       3   baBabab
```

If we define the generator order to be that of the columns, then the table above is not in canonic form, and the coset representatives are not in order. We now standardise the table; note the compaction phase before standardisation, although it does nothing in this particular case. Note how, if we read through the table in row-major order, new cosets are introduced using the smallest available number, and that the representatives are now in order and are minimal.

```
st;
co/ST: a=12 r=13 h=1 n=13; c=+0.00
```

```
pr:-1,12;
co: a=12 r=13 h=1 n=13; c=+0.00
 coset |      b      B      a   order   rep've
-------+-------------------------------------
     1 |      2      3      3
     2 |      3      1      4       3   b
     3 |      1      2      1       3   B
     4 |      5      6      2       5   ba
     5 |      6      4      7       5   bab
     6 |      4      5      8       2   baB
     7 |      8      9      5       5   baba
     8 |      9      7      6       5   baBa
     9 |      7      8     10       3   babaB
    10 |     11     12      9       3   babaBa
    11 |     12     10     12       3   babaBab
    12 |     10     11     11       2   babaBaB
```

We now exit ACE, printing out the version and the date & time again.

```
q
=========================================
ACE 3.001        Wed Apr  4 23:09:17 2001
```

A.2 Emulating Sims

Here we demonstrate the various `sims` modes, and see if we can duplicate the `m` (maximum active cosets) and `t` (total cosets defined) values (see the input file `ex001.in`). The ability to do so gives our confidence in the correctness of ACE a large boost. (In Section A.8, we show how we can use `standard` and one of ACE's `sims` modes to approximate one of Sims' even-numbered strategies.) We work with the formal inverses of the relators and subgroup generators from [15], since definitions are made from the front in Sims' routines and from the rear in ACE. We may also have to use the `asis` flag, to force the column order (by entering involutions as `xx`) and to preserve the relator ordering. We match Sims' values for R style & R* style (`sims:1` & 3) and C style (`sims:9`), but may not do so if we use Mendelsohn (`sims:5` & 7); this makes sense, since the order of processing cycled relators is not specified by Sims.

The input and output files for Example 5.2:

```
gr: r,s,t;
rel: (r^tRR)^-1, (s^rSS)^-1, (t^sTT)^-1;
text: ;                                 sr;
text: ** Sims:1 (cf. 1502/1550) ...;  sims:1;  end;
text: ** Sims:3 (cf. 673/673) ...;    sims:3;  end;
text: ** Sims:5 (cf. 1808/1864) ...;  sims:5;  end;
text: ** Sims:7 (cf. 620/620) ...;    sims:7;  end;
text: ** Sims:9 (cf. 588/588) ...;    sims:9;  end;
```

```
  #--- ACE 3.001: Run Parameters ---
Group Name: G;
Group Relators: rrTRt, ssRSr, ttSTs;
Subgroup Name: H;
Subgroup Generators: ;
  #---------------------------------
```

```
** Sims:1 (cf. 1502/1550) ...
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.00; m=1502 t=1550)
** Sims:3 (cf. 673/673) ...
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.00; m=673 t=673)
** Sims:5 (cf. 1808/1864) ...
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.01; m=1603 t=1603)
** Sims:7 (cf. 620/620) ...
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.00; m=615 t=615)
** Sims:9 (cf. 588/588) ...
INDEX = 1 (a=1 r=2 h=2 n=2; l=4 c=0.00; m=588 t=588)
```

The input and output files for Example 5.3, $k = 8$:

```
gr: x,y;
rel: (xx)^-1, (y^3)^-1, ((xy)^7)^-1, ((xyxY)^8)^-1;
text: ;                                sr;
text: ** Sims:1 (cf. 87254/128562) ...;  sims:1;  end;
text: ** Sims:3 (cf. 31678/32320) ...;   sims:3;  end;
text: ** Sims:5 (cf. 99632/178620) ...;  sims:5;  end;
text: ** Sims:7 (cf. 30108/31365) ...;   sims:7;  end;
text: ** Sims:9 (cf. 39745/39745) ...;   asis:1;  sims:9;  end;
```

```
  #--- ACE 3.001: Run Parameters ---
Group Name: G;
Group Relators: XX, YYY, YXYXYXYXYXYXYX, yXYXyXYXyXYXyXYXyXYXyXYXyXYXyXYX;
Subgroup Name: H;
Subgroup Generators: ;
  #---------------------------------
** Sims:1 (cf. 87254/128562) ...
INDEX = 10752 (a=10752 r=128563 h=1 n=128563; l=27 c=0.16; m=87254 t=128562)
** Sims:3 (cf. 31678/32320) ...
INDEX = 10752 (a=10752 r=8005 h=32321 n=32321; l=10 c=0.13; m=31678 t=32320)
** Sims:5 (cf. 99632/178620) ...
INDEX = 10752 (a=10752 r=168547 h=1 n=168547; l=24 c=0.24; m=96952 t=168546)
** Sims:7 (cf. 30108/31365) ...
INDEX = 10752 (a=10752 r=5738 h=31673 n=31673; l=8 c=0.14; m=30420 t=31672)
** Sims:9 (cf. 39745/39745) ...
INDEX = 10752 (a=10752 r=1 h=39746 n=39746; l=43 c=0.19; m=39745 t=39745)
```

The input and output files for Example 5.4:

```
gr: a,b;
rel: (a^8)^-1, (b^7)^-1, ((ab)^2)^-1, ((Ab)^3)^-1;
gen: (a^2)^-1, (Ab)^-1;
asis:1;
text: ;                               sr;
text: ** Sims:1 (cf. 2174/2635) ...;  sims:1;  end;
text: ** Sims:3 (cf. 1199/1212) ...;  sims:3;  end;
text: ** Sims:5 (cf. 2213/2619) ...;  sims:5;  end;
text: ** Sims:7 (cf. 1258/1284) ...;  sims:7;  end;
text: ** Sims:9 (cf. 1302/1306) ...;  asis:0;  sims:9;  end;
```

```
  #--- ACE 3.001: Run Parameters ---
Group Name: G;
Group Relators: AAAAAAAA, BBBBBBB, BABA, BaBaBa;
Subgroup Name: H;
Subgroup Generators: AA, Ba;
  #---------------------------------
```

```
** Sims:1 (cf. 2174/2635) ...
INDEX = 448 (a=448 r=2636 h=1 n=2636; l=4 c=0.00; m=2174 t=2635)
** Sims:3 (cf. 1199/1212) ...
INDEX = 448 (a=448 r=576 h=1213 n=1213; l=3 c=0.01; m=1199 t=1212)
** Sims:5 (cf. 2213/2619) ...
INDEX = 448 (a=448 r=2620 h=1 n=2620; l=4 c=0.00; m=2213 t=2619)
** Sims:7 (cf. 1258/1284) ...
INDEX = 448 (a=448 r=612 h=1285 n=1285; l=3 c=0.01; m=1258 t=1284)
** Sims:9 (cf. 1302/1306) ...
INDEX = 448 (a=448 r=1 h=1307 n=1307; l=5 c=0.00; m=1302 t=1306)
```

A.3 Row filling

If all definitions are made by applying cosets to relators, then the coset table can contain holes, either because the form of the relators 'hides' one of the generators from one of the cosets, or because one of the generators is not present in the relators. The `row` and `mend` parameters can be used to deal with these sorts of situations. Consider the following examples, drawn from [17]; see the input file `ex002.in`. Note that, although the `row` parameter is specifically intended to prevent the table containing holes, the `mend` parameter actually yields better enumeration statistics. Note also the use of the `asis` parameter to control whether or not the presentation is reduced.

```
enum:infinite cyclic group; gr:xy;   rel:yxyxY;
subgr:self (index 1);       gen:x;
asis:1;  mess:1000000;      pure r;  end;
  #--- ACE 3.001: Run Parameters ---
Group Name: infinite cyclic group;
Group Generators: xy;
Group Relators: yxyxY;
Subgroup Name: self (index 1);
Subgroup Generators: x;
Wo:1000000; Max:249998; Mess:1000000; Ti:-1; Ho:-1; Loop:0;
As:1; Path:0; Row:0; Mend:0; No:0; Look:0; Com:100;
C:0; R:1000; Fi:1; PMod:0; PSiz:256; DMod:0; DSiz:1000;
  #---------------------------------
SG: a=1 r=1 h=1 n=2; l=1 c=+0.00; m=1 t=1
OVERFLOW (a=249992 r=249996 h=1 n=249999; l=253 c=0.19; m=249992 t=249998)
pr:-1,12;
CO: a=249992 r=249990 h=1 n=249993; c=+0.05
 coset |      x      X      y      Y   order   rep've
-------+---------------------------------------------
     1 |      1      1      2      0
     2 |      4      3      5      1       0   y
     3 |      2      5      6      4       0   yX
     4 |      0      2      3      0       0   yx
     5 |      3      6      7      2       0   yy
     6 |      5      7      8      3       0   yXy
     7 |      6      8      9      5       0   yyy
     8 |      7      9     10      6       0   yXyy
     9 |      8     10     11      7       0   yyyy
    10 |      9     11     12      8       0   yXyyy
    11 |     10     12     13      9       0   yyyyy
    12 |     11     13     14     10       0   yXyyyy
```

```
mess:0;
pure r;  row:1;  end;
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.00; m=12 t=17)
pure r;  mend:1;  end;
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.00; m=5 t=6)

mess:1000000;
asis:0;  pure r;  end;
  #--- ACE 3.001: Run Parameters ---
Group Name: infinite cyclic group;
Group Generators: xy;
Group Relators: xyx;
Subgroup Name: self (index 1);
Subgroup Generators: x;
Wo:1000000; Max:249998; Mess:1000000; Ti:-1; Ho:-1; Loop:0;
As:0; Path:0; Row:0; Mend:0; No:0; Look:0; Com:100;
C:0; R:1000; Fi:1; PMod:0; PSiz:256; DMod:0; DSiz:1000;
  #---------------------------------
SG: a=1 r=1 h=1 n=2; l=1 c=+0.00; m=1 t=1
UH: a=1 r=2 h=2 n=2; l=3 c=+0.00; m=1 t=1
INDEX = 1 (a=1 r=2 h=2 n=2; l=3 c=0.00; m=1 t=1)

enum:C_3;  rel:x^3yxyX^3,y^3xyxY^3;  subgr:trivial (index 3);  gen:;
asis:1;    pure r;                   end;
  #--- ACE 3.001: Run Parameters ---
Group Name: C_3;
Group Generators: xy;
Group Relators: xxxyxyXXX, yyyxyxYYY;
Subgroup Name: trivial (index 3);
Subgroup Generators: ;
Wo:1000000; Max:249998; Mess:1000000; Ti:-1; Ho:-1; Loop:0;
As:1; Path:0; Row:0; Mend:0; No:0; Look:0; Com:100;
C:0; R:1000; Fi:1; PMod:0; PSiz:256; DMod:0; DSiz:1000;
  #---------------------------------
OVERFLOW (a=181146 r=38770 h=1 n=249999; l=32 c=0.12; m=181146 t=249998)
pr:-1,16;
CO: a=181146 r=28074 h=1 n=181147; c=+0.03
 coset |      x      X      y      Y   order   rep've
-------+--------------------------------------------
     1 |      2      0      7      0
     2 |      3      1     15      0       0   x
     3 |      4      2     23      0       0   xx
     4 |     12      3      6      5       0   xxx
     5 |     35      6      4      0       0   xxxY
     6 |      5      0     31      4       0   xxxy
     7 |     47      0      8      1       0   y
     8 |     55      0      9      7       0   yy
     9 |     11     10     52      8       0   yyy
    10 |      9      0     72     11       0   yyyX
    11 |     63      9     10      0       0   yyyx
    12 |     20      4     14     13       0   xxxx
    13 |     89     14     12      0       0   xxxxY
    14 |     13      0     85     12       0   xxxxy
    15 |    101      0     16      2       0   xy
    16 |    109      0     17     15       0   xyy
```

```
mess:0;
pure r;  row:1;   end;
INDEX = 3 (a=3 r=468 h=1 n=468; l=3 c=0.00; m=343 t=467)
pure r;  mend:1;  end;
INDEX = 3 (a=3 r=29 h=29 n=29; l=3 c=0.00; m=21 t=28)

mess:1000000;
asis:0;  pure r;  end;
  #--- ACE 3.001: Run Parameters ---
Group Name: C_3;
Group Generators: xy;
Group Relators: yxy, xyx;
Subgroup Name: trivial (index 3);
Subgroup Generators: ;
Wo:1000000; Max:249998; Mess:1000000; Ti:-1; Ho:-1; Loop:0;
As:0; Path:0; Row:0; Mend:0; No:0; Look:0; Com:100;
C:0; R:1000; Fi:1; PMod:0; PSiz:256; DMod:0; DSiz:1000;
  #---------------------------------
UH: a=3 r=6 h=6 n=6; l=3 c=+0.00; m=5 t=5
INDEX = 3 (a=3 r=6 h=6 n=6; l=3 c=0.01; m=5 t=5)
```

A.4 Equivalent presentations

TBA: $F(2, 7)$, using `rep` & `aep` ...

A.5 Deduction queues

TBA: ... (see test009)

A.6 Large enumerations

Suppose that the presentation given is such that the final coset table exceeds the 4 Gbyte limit imposed by 32-bit machines; e.g., an index of 250×10^6, with a 5-column table and 4 byte integers. We are justified in regarding such an enumeration as 'big', since it will require more than 4 Gbyte of storage no matter how efficiently it is performed. Of course, even trivial enumerations may exceed this limit if they are very pathological (see, e.g., [7]). However, we have no (easy) way of knowing whether or not such enumerations can be done within the 4 Gbyte limit, so we are hesitant to classify them as big. ACE is 64-bit 'aware', and can use more than 4 Gbyte of memory if it is available. Note however that the number of cosets (i.e., the number of rows in the coset table) is still limited by the size of a signed `int`. So the maximum size of a table is $2^{31} - n$ cosets, where n is probably 3; one since we can't actually represent +2147483648, one since coset #0 is not used, and one since we need to count one past the top of the table.

Some trivial group enumerations involving more than 1 G total cosets and 4 Gbyte of memory were reported in [7]. However, the first big enumeration, in the above sense, done by ACE was the Thomson simple group. This group has order TBA, and contains TBA as an index TBA subgroup. TBA: ...

A.7 Looping

TBA: ...

A.8 Use of st

The following shows how we can approximate one of the even-numbered Sims strategies by repeatedly pausing ACE, standardising and continuing. Below, we use restrictive values of `max` to pause ACE, starting with `max:14` and stepping `max` by `50` until the enumeration completes. It is easy to create the loop necessary to do this within some higher level programming language such as GAP that can interface with ACE. Recall, from Section A.2, that definitions are made from the front in Sims' routines and from the rear in ACE; so we work with the formal inverses of the relators and subgroup generators from [15]. As it happens we are able to generate `t` and `m` statistics for Example 5.2 with strategy 4 that exactly match those of Sims [15, Table 5.5.3]. The example that does so is `ex007.in`. Here now is that input file and its output.

```
gr: r,s,t;
rel: (r^tRR)^-1, (s^rSS)^-1, (t^sTT)^-1;
text: ;                               sr;
text: ** Sims:4 (cf. 393/393) ...;
sims:3;
max:14;
Start;
standard; max:64;  Continue; standard; max:114; Continue;
standard; max:164; Continue; standard; max:214; Continue;
standard; max:264; Continue; standard; max:314; Continue;
standard; max:364; Continue; standard; max:414; Continue;
```

```
  #--- ACE 3.001: Run Parameters ---
Group Name: G;
Group Relators: rrTRt, ssRSr, ttSTs;
Subgroup Name: H;
Subgroup Generators: ;
  #---------------------------------
** Sims:4 (cf. 393/393) ...
OVERFLOW (a=14 r=2 h=2 n=15; l=3 c=0.00; m=14 t=14)
co/ST: a=14 r=2 h=2 n=15; c=+0.00
OVERFLOW (a=64 r=8 h=8 n=65; l=2 c=0.00; m=64 t=64)
co/ST: a=64 r=8 h=8 n=65; c=+0.00
OVERFLOW (a=114 r=15 h=15 n=115; l=2 c=0.00; m=114 t=114)
co/ST: a=114 r=15 h=15 n=115; c=+0.00
OVERFLOW (a=164 r=23 h=23 n=165; l=2 c=0.00; m=164 t=164)
co/ST: a=164 r=23 h=23 n=165; c=+0.00
OVERFLOW (a=214 r=31 h=31 n=215; l=2 c=0.00; m=214 t=214)
co/ST: a=214 r=31 h=31 n=215; c=+0.00
OVERFLOW (a=264 r=39 h=39 n=265; l=2 c=0.00; m=264 t=264)
co/ST: a=264 r=39 h=39 n=265; c=+0.00
OVERFLOW (a=314 r=47 h=47 n=315; l=2 c=0.00; m=314 t=314)
co/ST: a=314 r=47 h=47 n=315; c=+0.00
OVERFLOW (a=364 r=56 h=56 n=365; l=2 c=0.00; m=364 t=364)
co/ST: a=364 r=56 h=56 n=365; c=+0.00
INDEX = 1 (a=1 r=2 h=2 n=2; l=2 c=0.00; m=393 t=393)
```

A.9 Use of `cy`, `nc`, `cc` and `rc`

TBA: ...

Appendix B

Command summary

This appendix gives details of all the commands available when using the interactive interface. The section headings match the help screen produced by the help command, and are in the same order. Alternative forms of a command are separated by a /, while any optional part of a command is denoted by [...]. Case is not significant in command names, but that part of a command actually present must be correct, modulo white space. Appendix A contains many examples of how to correctly drive ACE.

Parameters to a command are supplied after a colon (:). Each command is terminated by a newline or a semicolon (;), except in some cases where the argument may be a list of words, in which case newlines are ignored and a semicolon is the only terminator. (E.g., the add gen, add rel, rel & gen commands.) In many cases the parameters are optional, and entering the command without an argument prints the parameter's current value. If the no-argument form has a special meaning, this is noted in its entry below. Where there is no danger of confusion, the : and/or the ; can usually be dispensed with. The allowed parameter values are listed after a colon (:) either explicitly (e.g. 1..7 means an integer in the range 1 to 7 inclusive) or is one of the following:

<int>	an integer;
<int list>	a comma-separated list of <int>;
<string>	an alphanumeric string (blanks allowed, but no semicolons);
<filename>	a <string> (but must be a valid UNIX filename);
<letter list>	a list of lower-case letters, optionally separated by blanks or commas;
<word list>	a comma-separated list of <word>s;
<relator list>	a comma-separated list of <word>s or <relation>s;

where a <relation> is an equals (=) separated list of <word>s, a (somewhat sketchy) BNF for <word> is given by

```
<word>    = <factor> { "*" | "/" <factor> }
<factor>  = <element> [ ["^"] <integer> | "^" <element> ]
<element> = <generator> ["'"]
            | "(" <word> { "," <word> } ")" ["'"]
            | "[" <word> { "," <word> } "]" ["'"]
```

and a <generator> is a letter or an integer (see B.27). A verbal description of a <word list> is given in B.28.

NOTE: Some of the command names or synonyms might strike you as peculiar. These names were *not* chosen by me, but were dictated by the need for compatibility with previous coset enumerators (ie, `tc` & `ce/ace`).

B.1 `add gen[erators] / sg : <word list> ;`

Adds the words in the list to any subgroup generators already present. The enumeration must be (re)started or redone, it cannot be continued.

B.2 `add rel[ators] / rl : <relation list> ;`

Adds the words in the list to any relators already present. The enumeration must be (re)started or redone, it cannot be continued.

B.3 `aep : 1..7 ;`

The `aep` (all equivalent presentations) option runs an enumeration for each possible combination of relator ordering, relator rotations, and relator inversions. As discussed by Cannon, Dimino, Havas & Watson [3] and Havas & Ramsay [8] such equivalent presentations can yield large variations in the number of cosets required in an enumeration. For this command, we are interested in this variation.

The `aep` option first performs a 'priming run' using the parameters as they stand. In particular, the `asis` & `mess` parameters are honoured. It then turns `asis` on and `mess` off, and generates and tests the requested equivalent presentations. The maximum and minimum values attained by `maxcos` & `totcos` are tracked, and each time a new 'record' is found the relators used and the summary result line is printed. At the end, some additional status information is printed: the number of runs which yielded a finite index; the total number of runs (excluding the priming run); and the range of values observed for `maxcos` & `totcos`. `asis` & `mess` are now restored to their original values, and the system is ready for further commands.

The mandatory argument is considered as a binary number. Its three bits are treated as flags, and control relator rotations (the 2^0 bit), relator inversions (the 2^1 bit) and relator orderings (the 2^2 bit); "1" means 'active' and "0" means 'inactive'. The order in which the equivalent presentations are generated and tested has no particular significance, but note that the presentation as given (*after* the initial priming run) is the *last* presentation to be generated and tested, so that the group's relators are left 'unchanged' by running the `aep` option.

As an example (drawn from the discussion in [8]) consider the index 448 enumeration of $(8,7 \mid 2,3)/\langle a^2, Ab\rangle$, where

$$(8,7 \mid 2,3) = \langle a, b \mid a^8 = b^7 = (ab)^2 = (Ab)^3 = 1\rangle.$$

There are $4! = 24$ relator orderings and $2^4 = 16$ combinations of relator or inverted relator. Exponents are taken into account when rotating relators, so the relators given give rise to 1, 1, 2 & 2 rotations respectively, for a total of $1.1.2.2 = 4$ combinations. So the command `aep:7` would generate and test $24.16.4 = 1536$ equivalent presentations, while `aep:3` would generate and test $16.4 = 64$ equivalent presentations.

NOTES: There is no way to stop the `aep` option before it has completed, other than killing the task. So do a reality check beforehand on the size of the search space and the time for each enumeration. If you are interested in finding a 'good' enumeration, it can be very helpful, in terms of running time, to put a tight limit on the number of cosets via the `max` parameter. (You may also have to set `com:100` to prevent time-wasting attempts to recover space via compaction.) This maximises throughput by causing the 'bad' enumerations, which are in the majority, to overflow quickly and abort. If you wish to explore a very large search-space, consider firing up many copies of ACE, and starting each with a 'random' equivalent presentation. Alternatively, you could use the `rep` command.

B.4 `ai / alter i[nput] : [<filename>] ;`

By default, commands to ACE are read from the standard input file (i.e., the 'keyboard', `stdin`). The `ai` command closes the current input file (unless it's `stdin`), and opens `<filename>` as the source of commands. If `<filename>` can't be opened, input reverts to `stdin`.

NOTES: If you switch to taking input from a(nother) file, remember to switch back to the previous file before the end of the current file. If you don't, the `EOF` in the current file will cause ACE to terminate.

B.5 `ao / alter o[utput] : [<filename>] ;`

By default, output from ACE is sent to the standard output file (i.e., the 'terminal', `stdout`). The `ao` command closes the current output file, and opens `<filename>` for all future output. If `<filename>` can't be opened, output reverts to `stdout`.

B.6 `as[is] : [0/1] ;`

By default, ACE freely & cyclically reduces the relators, freely reduces the subgroup generators, and sorts relators & generators in length-increasing order (a stable insertion sort is used). If you do not want this, you can switch it off by `asis:1`.

NOTES: As well as allowing you to process the presentation in the form given, this is useful for forcing definitions to be made in a prespecified order, by introducing dummy (i.e., freely trivial) subgroup generators. Note also that the exact form of the presentation can have a significant impact on the enumeration statistics; it is not the case that the default option always yields the best enumeration.

GURU NOTES: When `asis:0`, a (reduced) relator of the form `aa` or `AA` causes that generator to be treated as an involution. In the relators and subgroup generators, the inverses of involutionary generators are automatically replaced with the generator.

When `asis:1`, only relators of the form `a^2` cause the generator to be treated as an involution. The forms `aa` & `a^2` are preserved in any printout, so that you can track ACE's behaviour.

B.7 `beg[in] / end / start ;`

Start an enumeration. Any existing information in the table is cleared, and the enumeration starts from coset #1 (i.e., the subgroup).

B.8 `bye / exit / q[uit] ;`

This exits ACE nicely, printing the date and the time. An `EOF` (end-of-file; i.e., `^d`) has the same effect, so proper termination occurs if ACE is taking its input from a script file.

B.9 `cc / coset coinc[idence] : <int> ;`

Print out the representative of coset #`<int>`, and add it to the subgroup generators; i.e., equates this coset with coset #1, the subgroup.

B.10 `c[factor] / ct[ factor] : [<int>] ;`

The value of this parameter sets the 'blocking factor' for C-style definitions; i.e., the number of coset definitions made by filling the next empty coset table position during each pass through the enumerator's main loop. The absolute value of `<int>` is the value used. The enumeration style is selected by the values of the `ct` & `rt` parameters; see Section 3.1.

B.11 `check / redo ;`

An extant enumeration is redone, using the current parameters. Any existing information in the table is retained, and the enumeration is restarted from coset #1 (i.e., the subgroup).

NOTES: This option is really intended for the case where additional relators and/or subgroup generators have been introduced. The current table, which may be incomplete or exhibit a finite index, is still 'valid'. However, the additional data may allow the enumeration to complete, or cause a collapse to a smaller index.

B.12 `com[paction] : [0..100] ;`

The key word `com` controls compaction of the coset table during an enumeration. Compaction recovers the space allocated to cosets which are flagged as dead (i.e., which were found to be coincident with lower-numbered cosets). It results in a table where all the active cosets are numbered contiguously from #1, and with the remainder of the table available for new cosets.

The coset table is compacted when a coset definition is required, there is no space for a new coset available, and provided that the given percentage of the coset table contains dead cosets. For example, `com:20` means compaction will occur only if 20% or more of the cosets in the table are dead. The argument can be any integer in the range 0–100, and the default value is 10 or 100; see Section 3.2. An argument of 100 means that compaction is never performed, while an argument of 0 means always compact, no matter how few dead cosets there are (provided there is at least one, of course).

Compaction may be performed multiple times during an enumeration, and the table that results from an enumeration may or may not be compact, depending on whether or not there have been any coincidences since the last compaction (or from the start of the enumeration, if there have been no compactions). If messaging is enabled (i.e., `mess` $\neq 0$), then a progress message (labelled `CO`) is printed each time the compaction routine is actually called (as opposed to each time it is potentially called).

NOTES: In some strategies (e.g., HLT) a lookahead phase may be run before compaction is attempted. In other strategies (e.g., `sims:3`) compaction may be performed while there are outstanding deductions; since deductions are discarded during compaction, a final `RA` phase will (automatically) be performed. Compacting a table 'destroys' information and history, in the sense that the table entries for any dead cosets are deleted, along with their coincidence list data. At Level 2, it is not possible to access the 'data' in dead cosets; in fact, most options that require table data compact the table automatically before they run.

B.13 `cont[inue] ;`

Continue the current enumeration, building upon the existing table. If a previous run stopped without producing a finite index you can, in principle, change any of the parameters and continue on. Of course, if you make any changes which invalidate the current table, you won't be allowed to continue, although you may be allowed to redo.

B.14 `cy[cles] ;`

Print out the table in cycles; i.e., the permutation representation.

B.15 `ded mo[de] / dmod[e] : [0..4] ;`

A completed table is only valid if every table entry has been tested in all essentially different positions in all relators. This testing can either be done directly (Felsch strategy) or via relator scanning (HLT strategy). If it is done directly, then more than one deduction (i.e., table entry) can be outstanding at any one time. So the untested deductions are stored in a stack. Normally this stack is fairly small but, during a collapse, it can become very large.

This command allows the user to specify how deductions should be handled. The options are: `0`, discard deductions if there is no stack space left; `1`, as `0`, but purge redundant cosets off the top of the stack at every coincidence; `2`, as `0`, but purge all redundant cosets from the stack at every coincidence; `3`, discard the entire stack if it overflows; `4`, if the stack overflows, then double the stack size and purge all redundant cosets from the stack.

The default deduction mode is either 0 or 4, depending on the strategy selected (see Section 3.2), and it is recommended that you stay with the default. If you want to know more details, read the code.

NOTES: If deductions are discarded for any reason, then a final `RA` phase will be run automatically at the end of the enumeration, if necessary, to check the result.

B.16 `ded si[ze] / dsiz[e] : [0/1..] ;`

Sets the size of the (initial) allocation for the deduction stack. The size is in terms of the number of deductions, with one deduction taking two words (i.e., 8 bytes). The default size, of 1000, can be selected by a `0` argument. See the `dmod` entry for a (brief) discussion of deduction handling.

B.17 `def[ault] ;`

This selects the default strategy, which is based on the defaulted R/C style; see Sections 3.1 & 3.2. The idea here is that we assume that the enumeration is 'easy', and start out in R style. If it turns out not to be easy, then we regard it as 'hard', and switch to CR style, after performing a lookahead on the entire table.

B.18 `del gen[erators] / ds : <int list> ;`

This command allows subgroup generators to be deleted from the presentation. If the generators are numbered from one in the output of, say, the `sr` command, then the generators listed in `<int list>` are deleted. `<int list>` must be a strictly increasing sequence.

B.19 `del rel[ators] / dr : <int list> ;`

As `del gen`, but for the group's relators.

B.20 `d[ump] : [0/1/2[,0/1]] ;`

Dumps the internal variables of ACE. The first argument selects which of the three levels of ACE to dump, and defaults to Level 0. The second argument selects the level of detail, and defaults to 0 (i.e., less detail). This option is intended for gurus; the source code should be consulted to see what the output means.

B.21 `easy ;`

If this strategy is selected, we run in R style (i.e., HLT) and turn lookahead & compaction off. For small and/or easy enumerations, this mode is likely to be the fastest.

B.22 `echo : [0/1] ;`

By default, ACE does not echo its commands. If you wish it to do so, turn this feature on with `echo:1`. This feature can be used to render output files from ACE less incomprehensible.

B.23 `enum[eration] / group name : <string> ;`

This command defines the name by which the current enumeration (i.e., the group being used) will be identified in any printout. It has no effect on the actual enumeration, and defaults to `G`. An empty name is accepted; to see what the current name is, use the `sr` command.

B.24 `fel[sch] : [0/1] ;`

An argument of `0` or no argument selects the Felsch strategy, while an argument of `1` selects Felsch with all relators in the subgroup and turns gap-filling on; see Section 3.2.

B.25 `f[factor] / fi[ll factor] : [0/1..] ;`

If gap-filling is on, then gaps of length one found during deduction testing are preferentially filled (see [6]). However, this potentially violates the formal requirement that all rows in the table are eventually filled (and tested against the relators). The fill factor is used to ensure that some constant proportion of the coset table is always kept filled. Before defining a coset to fill a gap of length one, the enumerator checks whether `fi` times the completed part of the table is at least the total size of the table and, if not, fills rows in standard order instead of the gap.

An argument of `0` selects the default value of $\lfloor 5(n+2)/4 \rfloor$, where n is the number of columns in the table. All other things being equal, we'd expect the ratio of the total size of the table to the completed part of the table to be $n+1$, so the default fill factor allows a moderate amount of gap-filling.

NOTES: If `fi` is smaller than n, then there is generally no gap-filling, although its processing overhead is still incurred. A large value of `fi` can cause infinite looping. However, in general, a large value does work well. The effects of the various gap-filling strategies vary widely. It is not clear which values are good general defaults or, indeed, whether any strategy is always 'not too bad'.

B.26 gen[erators] / subgroup gen[erators] : ¡word list¿ ;

By default, there are no subgroup generators and the subgroup is trivial. This command allows a list of subgroup generating words to be entered. The format is the same as for relators, except that 'genuine' relations (i.e., $w_1 = w_2$) are not allowed.

B.27 gr[oup generators]: [<letter list> / <int>] ;

This command introduces the group generators, which may be represented in one of two ways. They may be presented as a list of lower-case letters, optionally separated by commas. This is the usual method, and gives up to 26 generators. Subsequently, upper-case letters can be used, if desired, to stand for the inverse of their lower-case versions; e.g., A for a^-1, B for b^-1, etc. Alternatively, a positive integer can be used to indicate the number of generators. For example, gr:5 indicates that there are five generators, designated 1, 2, 3, 4 & 5, with inverses -1, etc.

NOTES: Any use of the gr command which actually defines generators invalidates any previous enumeration, and stays in effect until the next gr command. Any words for the group or subgroup must be entered using the nominated generator format, and all printout will use this format. This command is not optional, nor is there any default. A valid set of generators is the minimum information necessary before ACE will attempt an enumeration.

GURU NOTES: The columns of the coset table are allocated in the same order as the generators are listed, insofar as this is possible, given that the first two columns must be a generator/inverse pair or a pair of involutions. (This is an implementation issue, and is not formally necessary; see [1].) The ordering of the columns can, in some cases, affect the definition sequence of cosets and impact the statistics of an enumeration.

B.28 group relators / rel[ators] : <relation list> ;

By default, or if an empty argument to this command is used, the group is free. Otherwise, this command is used to introduce the group's defining relators. In order to allow ACE to accept presentations from a variety of sources, many kinds of word representations are allowed. ACE accepts words in the nominated generators, allowing * for multiplication, ^ for exponentiation and conjugation, and brackets for precedence specification. Round or square brackets may be used for commutation. (There is no danger of confusion between [a,b]/(a,b) and (ab), since a , implies commutation, while no comma implies a word.) If letter generators are used, multiplication and exponentiation signs (but *not* conjugation signs) may be omitted; e.g., a3 is the same as a^3 and ab is the same as a*b. Also, the exponent -1 can be abbreviated to -, so a- stands for A. Inverses can also be denoted by ' or /, so w_1w_2' $= w_1/w_2 = w_1w_2^{-1}$. The * can also be dropped for numeric generators; but of course two numeric generators, or a numeric exponent and a numeric generator, must be separated by whitespace.

Remember that `A` stands for `a^-1`, `a^b` for `Bab` and `[a,b]` & `[a,b,c]` for `ABab` & `[[a,b],c]`.

`<relation list>` is a comma-separated list of words (relators) or relations. A relation is a list of equated words, e.g. $w_1 = w_2 = w_3$ (equivalent to the relators $w_1 w_2^{-1}$ and $w_1 w_3^{-1}$).

B.29 `hard ;`

In many 'hard' enumerations, a mixture of R-style and C-style definitions, all tested in all essentially different positions, is appropriate. This option selects such a mixed strategy; see Section 3.2. The idea here is that most of the work is done C-style (with the relators in the subgroup and with gap-filling active), but that every 1000 C-style definitions a further coset is applied to all relators.

GURU NOTES: The `1000/1` mix is not necessarily optimal, and some experimentation may be needed to find an acceptable balance (see, for example, [8]). Note also that, the longer the total length of the presentation, the more work needs to be done for each coset application to the relators; one coset application can result in more than 1000 definitions, reversing the balance between R-style and C-style definitions.

B.30 `h[elp] ;`

Prints the help screen; i.e., all the headings in this appendix. Note that this list is fairly long, so its top may disappear off the top of the screen.

B.31 `hlt ;`

Selects the standard HLT strategy; see Section 3.2. Note that ACE's `hlt` has `lookahead` on; however, the sequence `hlt;lookahead:0;` easily achieves an HLT strategy with `lookahead` off.

B.32 `ho[le limit] : [-1/0..100] ;`

An experimental feature which allows an enumeration to be terminated when the percentage of holes in the table exceeds a given value. In practice, calculating this is very expensive, and it tends to remain constant or decrease throughout an enumeration. So the feature doesn't seem very useful. The default value of `-1` turns this feature off. If you want more details, read the source code.

B.33 `look[ahead] : [0/1..4] ;`

Although HLT-style strategies are fast, they are local, in the sense that the implications of any definitions/deductions made while applying cosets may not become apparent until much later. One way to alleviate this problem is to perform lookaheads occasionally; that is, to test the information in the table, looking for deductions or coincidences. ACE can perform a lookahead when the table overflows, before the compaction routine is called. An argument of `0` disables lookahead. Lookahead can be done using the entire table or only that part of the table above the current coset, and it can be done R-style (scanning cosets from the beginning of relators) or C-style (testing all definitions in all essentially different positions). An argument of `1` does a partial table, R-style lookahead; `2` does all the table, C-style; `3` does all the table, R-style; and `4` does a partial table, C-style. The default is either `0` or `1`; see Section 3.2.

NOTES: A lookahead can do a significant amount of work, so this phase may take a long time. The value of `mend` is honoured during R-style lookaheads.

B.34 `loop[ limit] : [0/1..] ;`

The core enumerator is organised as a state machine, with each step performing an 'action' (i.e., lookahead, compaction) or a block of actions (i.e., |`ct`| coset definitions, |`rt`| coset applications). The number of passes through the main loop (i.e., steps) is counted, and the enumerator can make an early return when this count hits the value of `loop`. A value of `0`, the default, turns this feature off.

GURU NOTES: You can do lots of really neat things using this feature, but you need some understanding of the internals of ACE to get real benefit from it. Read the code!

B.35 `max[ cosets] : [0/2..] ;`

By default, all of the workspace is used, if necessary, in building the coset table. So the table size is an upper bound on how many cosets can be active at any one time. The `max` option allows a limit to be placed on how much of the physical table space is made available to the enumerator. Enough space for at least two cosets (i.e., the subgroup and one other) must be made available. An argument of `0` selects all of the workspace.

B.36 `mend[elsohn] : [0/1] ;`

Mendelsohn style processing during relator scanning/closing is turned on by `mend:1` and off by `mend:0`. Off is the default, and here coset applications are done only at the start (and end) of a relator. Mendelsohn on means that coset applications are done at all cyclic permutations of the (base) relator. The effect of the Mendelsohn parameter is case-specific. It can mean the difference between success or failure, or it can impact the number of cosets required, or it can have no effect on an enumeration's statistics.

NOTES: Processing all cyclic permutations of the relators can be very time-consuming, especially if the presentation if large. So, all other things being equal, the Mendelsohn flag should normally be left off. Note that Mendelsohn's paper [11] discusses tracing all cyclic shifts of both the relators and their formal inverses. ACE only process the relators. However, since relators are scanned from both the front and the rear, we effectively process the inverses.

Table B.1: Possible enumeration results

result	level	meaning
`INDEX = x`	0	finite index of `x` obtained
`OVERFLOW`	0	out of table space
`SG PHASE OVERFLOW`	0	out of space (processing subgroup generators)
`ITERATION LIMIT`	0	`loop` limit triggered
`TIME LIMT`	0	`ti` limit triggered
`HOLE LIMIT`	0	`ho` limit triggered
`INCOMPLETE TABLE`	0	all cosets applied, but table has holes
`MEMORY PROBLEM`	1	out of memory (building data structures)

B.37 `mess[ages] / mon[itor] : [<int>] ;`

By default, or if the argument is `0`, ACE prints out only a single line of information giving the result of each enumeration. If `mess` is non-zero then the presentation & the parameters are echoed at the start of the run, and messages on the enumeration's status as it progresses are also printed out. The absolute value of `<int>` sets the frequency of the progress messages, with a negative sign turning hole monitoring on. The initial printout of the presentation & the parameters is the same as that produced by the `sr:1` command; see Appendix A for some examples.

The result line gives the result of the call to the enumerator and some basic statistics (see Appendix A for some examples). The possible results are given in Table B.1; any result not listed represents an internal error and should be reported. The statistics given are, in order: `a`, number of active cosets; `r`, number of applied cosets; `h`, first (potentially) incomplete row; `n`, next coset definition number; `l`, number of main loop passes; `c`, total CPU time; `m`, maximum active cosets; and `t`, total cosets defined.

The progress message lines consist of an initial tag, some fixed statistics, and some variable statistics. The possible message tags are listed in Table B.2, along with their meanings. The tags indicate the function just completed by the enumerator. The tags with a 'y' in the 'action' column represent functions which are aggregated and counted. Every time this count overflows the value of `mess`, a message line is printed and the count is zeroed. Those tags flagged with a 'y*' are only present if the appropriate option has been included in the build (see the `opt` command). Tags with an 'n' in the 'action' column are not counted, and cause a message line to be output every time they occur. They also cause the action count to be reset.

The fixed portion of the statistics consists of the `a`, `r`, `h`, `n`, `l` & `c` values, as for the result line, except that `c` is the time since the previous message line. If `mess` < 0 then hole monitoring is active, and an `h` statistic (representing the percentage of holes in the table) is inserted between the `n` & `l` values. The variable portion of the statistics can be: the `m` & `t` values, as for the result line; `d`, the current size of the deduction stack; `s`, `d` & `c` (with `DS` tag), the new stack size, the non-redundant deductions retained, and the redundant deductions discarded.

Table B.2: Possible progress messages

message	action	meaning
`AD`	y	coset #1 application definition (`SG`/`RS` phase)
`RD`	y	R-style definition
`RF`	y	row-filling definition
`CG`	y	immediate gap-filling definition
`CC`	y*	coincidence processed
`DD`	y*	deduction processed
`CP`	y	preferred list gap-filling definition
`CD`	y	C-style definition
`Lx`	n	lookahead performed (type `x`)
`CO`	n	table compacted
`CL`	n	complete lookahead (table as deduction stack)
`UH`	n	updated completed-row counter
`RA`	n	remaining cosets applied to relators
`SG`	n	subgroup generator phase
`RS`	n	relators in subgroup phase
`DS`	n	stack overflowed (compacted and doubled)

Notes: Hole monitoring is expensive, so don't turn it on unless you really need it. If you wish to print out the presentation & the parameters, but not the progress messages, then set `mess` non-zero, but very large. (You'll still get the `SG`, `DS`, etc. messages, but not the `RD`, `DD`, etc. ones.) You can set `mess` to `1`, to monitor all enumerator actions, but be warned that this can yield very large output files.

B.38 `mo[de] ;`

Prints the possible enumeration modes, as deduced from the command history since the last call to the enumerator; see Section 3.1.

B.39 `nc / normal[ closure] : [0/1] ;`

This option takes the current table (which may or may not be complete), and traces $g^{-1}wg$ and gwg^{-1} for all group generators g and all subgroup generator words w. The trace starts at coset #1 (ie, the subgroup), and we note whether we get back to coset #1 or not. If we do not, then we print out a line of output. If the argument is present & set (ie, `1`), then the offending conjugate is also added to the subgroup generators; the default is not to do so. A single pass though the (original) subgroup generators is made, and scans which do *not* complete are *not* processed (ie, printed/added).

Notes: It is the *user's* responsibility to rerun the enumeration (& the `nc` option) as necessary until the situation stabilises.

B.40 `no[ relators in subgroup] : [-1/0/1..] ;`

It is sometimes helpful to include the relators in the subgroup, in the sense that they are applied to coset #1 at the start of an enumeration. An argument of `0` turns this feature off, and an argument of `-1` includes all the relators. A positive argument includes the appropriate number of relators, in order.

B.41 `oo / order[ option] : <int> ;`

This option finds a coset with order a multiple of |`<int>`| modulo the subgroup, and prints out its coset representative. If `<int>` <0, then all cosets meeting the requirement are printed. If `<int>` $=0$, then the orders of all cosets are printed. If `<int>` >0, then the first coset meeting the requirement is printed.

B.42 `opt[ions] ;`

This command dumps details of the options included in the version of ACE you're running; i.e., what compiler flags were set when the executable was built. A typical output, illustrating the default build, is:

```
ACE 3.001 executable built:
  Fri Mar 30 14:30:59 CEST 2001
Level 0 options:
  statistics package = on
  coinc processing messages = on
  dedn processing messages = on
Level 1 options:
  workspace multipliers = decimal
Level 2 options:
  host info = on
```

B.43 `par[ameters] ;`

An old option, which did nothing. It is included for backward comparability. Pre-ACE 3.001 scripts may contain this option, which is quietly ignored by ACE 3.001.

B.44 `path[ compression] : [0/1] ;`

To correctly process multiple coincidences, a union-find must be performed. If both path compression and weighted union are used, then this can be done in essentially linear time (see, e.g., [4]). Weighted union alone, in the worst-case, is worse than linear, but is subquadratic. In practice, path compression is expensive, since it involves many coset table accesses. So, by default, path compression is turned off; it can be turned on by `path:1`. It has no effect on the result, but may affect the running time and the internal statistics.

Guru Notes: The whole question of the best way to handle large coincidence forests is problematic. Formally, ACE does not do a weighted union, since it is constrained to replace the higher-numbered of a coincident pair. In practice, this seems to amount to much the same thing! Turning path compression on cuts down the amount of data movement during coincidence processing at the expense of having to trace the paths and compress them. In general, it does not seem to be worthwhile.

B.45 `pd mo[de] / pmod[e] : [0/1..3] ;`

If the argument is `0`, then Felsch style definitions are made using the next empty table slot. If the argument is non-zero, then gaps of length one found during relator scans in Felsch style are preferentially filled (subject to the value of `fi`). If the argument is `1`, they are filled immediately, and if it is `2`, the consequent deduction is also made immediately (of course, these are also put on the deduction stack). If the argument is `3`, then the gaps are noted in the preferred definition queue. Provided a live such gap survives (and no coincidence occurs, which causes the queue to be discarded) the next coset will be defined to fill the oldest gap of length one. The default value is either `0` or `3`, depending on the strategy selected (see Section 3.2). If you want to know more details, read the code.

B.46 `pd si[ze] / psiz[e] : [0/2/4/8/...] ;`

The preferred definition queue is implemented as a ring, dropping earliest entries. Its size *must* be 2^n, for some $n > 0$. An argument of `0` selects the default size of 256. Each queue slot takes two words (i.e., 8 bytes), and the queue can store up to $2^n - 1$ entries.

B.47 `print det[ails] / sr : [0/1..5] ;`

This command prints out details of the current presentation and parameters. No argument, or an argument of `0`, prints out the group & subgroup name, the group's relators and the subgroup's generators. If the argument is `1`, then the group generators and the current setting of the enumeration control parameters are also printed. (This printout is the same as that produced at the start of a run when messaging is on.) Arguments of `2` – `5` print out the current values of `enum`, `rel`, `subg` & `gen`, respectively. See Appendix A for some examples.

NOTES: The output is printed out in a form suitable for input, so that a record of a previous run can be used to replicate the run. Note that, due to the defaulting of some parameters and the special meaning attached to some values, a little care has to be taken in interpreting the parameters. If you wish to *exactly* duplicate a run, you should use the output of `sr` *after* the run completes.

B.48 `pr[int table] : [[-]<int>[,<int>[,<int>]]] ;`

Compact the table, and then print it out from the first to the second argument, in steps of the third argument. If the first argument is negative, then the orders (if available) and representatives of the cosets are printed also. The third argument defaults to one. The one-argument form is equivalent to the two-argument form with a first argument of `1` and the argument used as the second argument. The no-argument form prints the entire table, without orders or representatives.

B.49 `pure c[t] ;`

Sets the strategy to basic C-style (coset table based) – no compaction, no gap-filling, no relators in subgroup; see Section 3.2.

B.50 `pure r[t] ;`

Sets the strategy to basic R-style (relator based) – no Mendelsohn, no compaction, no lookahead, no row-filling; see Section 3.2.

B.51 `rc / random coinc[idences]: <int>[,<int>] ;`

This option attempts to find nontrivial subgroups with index a multiple of the first argument by repeatedly putting random cosets coincident with coset #1 and seeing what happens. If the first argument is `0` any *non-trivial* finite index is accepted, while if it's `1` *any* finite index will do. The starting coset table must be non-empty, but should not be complete. The second argument puts a limit on the number of attempts, with a default of eight. For each attempt, we repeatedly add random coset representatives to the subgroup and redo the enumeration. If the table becomes too small, the attempt is aborted, the original subgroup generators restored, the CT is recalculated, and another attempt made. If an attempt succeeds, then the new set of subgroup generators is retained.

Guru Notes: (i) A coset can have many different representatives. Consider running `st` before `rc`, to canonicise the table and the representatives. This makes the reps minimal; sadly, however, it will only do so for the first of a series of attempts. (ii) If a series of attempts to find a subgroup fails, consider running the enumeration with different parameters. Although `rc` is random, it is always working with the same coset table; changing the parameters will give a different table and hence a different set of reps.

B.52 `rec[over] / contig[uous] ;`

Invokes the compaction routine on the table to recover the space used by any dead cosets. A `CO` message line is printed if any cosets were recovered, and a `co` line if none were. This routine is called automatically if the `cy`, `nc`, `pr` or `st` options are invoked.

B.53 `rep : 1..7[,<int>] ;`

The `rep` (random equivalent presentations) option complements the `aep` option. It generates and tests some random equivalent presentations. The mandatory argument acts as for `aep`, while the optional second argument sets the number of presentations, with a default of eight.

The routine first turns `asis` on and `mess` off, and then generates and tests the requested equivalent presentations. For each presentation the relators used and the summary result line is printed. `asis` & `mess` are now restored to their original values, and the system is ready for further commands.

NOTES: The relator inversions & rotations are 'genuinely' random. The relator permuting is a little bit of a kludge, with the 'quality' of the permutations tending to improve with successive presentations. When the `rep` command completes, the presentation active is the *last* one generated.

GURU NOTE: It might appear that neglecting to restore the original presentation is an error. In fact, it is a useful feature! Suppose that the space of equivalent presentations is too large to exhaustively test. As noted in the entry for `aep`, we can start up multiple copies of `aep` at random points in the search-space. Manually generating 'random' equivalent presentations to serve as starting-points is tedious and error-prone. The `rep` option provides a simple solution; simply run `rep` before `aep`!

B.54 `restart ;`

An old option, included for backward compatibility. Use the `check/redo` option instead. Pre-ACE 3.001 scripts may contain this option, which is quietly ignored by ACE 3.001.

B.55 `r[factor] / rt[ factor] : [<int>] ;`

The value of this parameter sets the 'blocking factor' for R-style definitions; i.e., the number of cosets applied to all the relators during each pass through the enumerator's main loop. The absolute value of `<int>` is the value used. The enumeration style is selected by the values of the `ct` & `rt` parameters; see Section 3.1.

B.56 `row[ filling] : [0/1] ;`

When making HLT-style definitions, it is normal to scan each row of the table after its coset has been applied to all relators, and make definitions to fill any holes encountered. Failure to do so can cause even simple enumerations to overflow; see Section A.3. To turn row filling off, use `row:0`.

B.57 `sc / stabil[ising cosets] : <int> ;`

This option takes the current table (which may or may not be complete), and looks for (the requested number of) cosets which 'stabilise' the subgroup. A coset c stabilises the subgroup $\langle w_1, \ldots, w_s \rangle$ if $cw_j = c$ for all $1 \leq j \leq s$. If `<int>` > 0, the first `<int>` stabilising cosets found are printed. If `<int>` $= 0$, all of the stabilising cosets, plus their representatives, are printed. If `<int>` < 0, the first $|$`<int>`$|$ stabilising cosets, plus their representatives, are printed.

B.58 `sims : 1/3/5/7/9 ;`

In his book [15], Sims discusses ten standard enumeration strategies. These are effectively HLT without `lookahead` (with or without the `mend` parameter, and in R or R* style) and Felsch, all either with or without table standardisation as the enumeration proceeds. ACE does not implement table standardisation on an ongoing basis, although tables from an incomplete or paused enumeration can be standardised before the enumeration is continued. The other five strategies are implemented, and can be selected by this command. The argument matches the number given in [15, §5.5]; see Section 3.2 for the parameter settings. With care, it is often possible to duplicate the statistics given in [15]; some examples are given in Sections A.2 and A.8.

B.59 `st[andard table] ;`

This option compacts and then standardises the table (which may or may not be complete). That is, for a given ordering of the generators in the columns of the table, it produces the 'canonic' version of the current table. In such a table, a row-major scan encounters previously unseen cosets in (contiguous) numeric order; see Section A.1 for an example.

NOTES: (i) In a canonic table, the coset representatives are in length plus (column order) lexicographic order, and each is the minimum in this order. Further, they are a Schreier set (ie, each prefix of a rep is also a rep). (ii) See Sims [15] for a discussion of standardising tables, and what this achieves.

GURU NOTES: In half of the ten standard enumeration strategies of Sims [15], the table is standardised repeatedly. This is expensive computationally, but can result in fewer cosets being necessary. The effect of doing this can be investigated in ACE by (repeatedly) halting the enumeration, standardising the table, and continuing; see Section A.8 for an example.

B.60 `stat[istics] / stats ;`

If the statistics package is compiled into the code (which it is by default, see the `opt` command), then dump the statistics accumulated during the most recent enumeration. See Section A.1 for an example, and the `enum.c` source file for the meaning of the variables.

B.61 `style ;`

Prints the current enumeration style, as deduced from the current `Ct` & `Rt` parameters; see Section 3.1.

B.62 `subg[roup name] : <string> ;`

This command defines the name by which the current subgroup will be identified in any printout. It has no effect on the actual enumeration, and defaults to `H`. An empty name is accepted; to see what the current name is, use the `sr` command.

B.63 `sys[tem] : <string> ;`

Passes `<string>` to a shell, via the C library routine `system()`.

B.64 `text : <string> ;`

Just echoes `<string>`. This allows the output from a run driven by a script to be tarted up.

B.65 `ti[me limit] : [-1/0/1..] ;`

The `ti` command puts a time limit (in seconds) on the length of a run. An argument of $<$ `0` mean there is no limit (the default). If the argument is $\geq$ `0` then the total elapsed time for this call is checked at the end of each pass through the enumerator's main loop, and if it's more than the limit the run is stopped and the current table returned. Note that a limit of `0` performs exactly one pass through the main loop, since `0` $\geq$ `0`. If the enumerator is run in the continue mode, this allows a form of 'single-stepping'. The time limit is approximate, in the sense that the enumerator may run for a longer, but never a shorter, time. So, if there is, e.g., a big collapse (so that the time round the loop becomes very long), then the run may run over the limit by a large amount.

NOTES: The time limit is CPU-time, not wall-time. As in all timing under Unix, the clock's granularity (usually 10 mSec) and the system load can affect the timing; so the number of main loop iterations in a given time may vary. If you want more precise control, use the `loop` option.

B.66 `tw / trace[ word] : <int>,<word> ;`

Traces `<word>` through the coset table, starting at coset `<int>`. Prints the final coset, if the trace completes.

B.67 `wo[rkspace] : [<int>[k/m/g]] ;`

By default, ACE has a physical table size of 10^6 entries (i.e., 4×10^6 bytes in the default 32-bit environment). The number of cosets in the table is the table size divided by the number of columns. The `wo` command allows the physical table size, in entries, to be set. The argument is multiplied by 1, 10^3, 10^6, or 10^9, depending as nothing, a `k`, an `m`, or a `g` is appended to the argument. Although the number of cosets is limited to $2^{31}-1$ (if the C `<int>` type is 32 bits), the table size can exceed the 4GByte 32-bit limit if a suitable machine is used.

NOTES: If the binary option is set (see the `opt` command), the multipliers are 1, 2^{10}, 2^{20} & 2^{30} respectively. The actual number of cosets in the table is entries/columns-2, rounded down to the nearest integer. The -2 is to allow for possible rounding errors and the fact that coset #0 is not used.

B.68 `# ... <newline>`

Any input between a sharp sign (`#`) and the next newline is ignored. This allows comments to be included anywhere in command scripts.

Appendix C

State machine details

Figure C.1: The R/C style

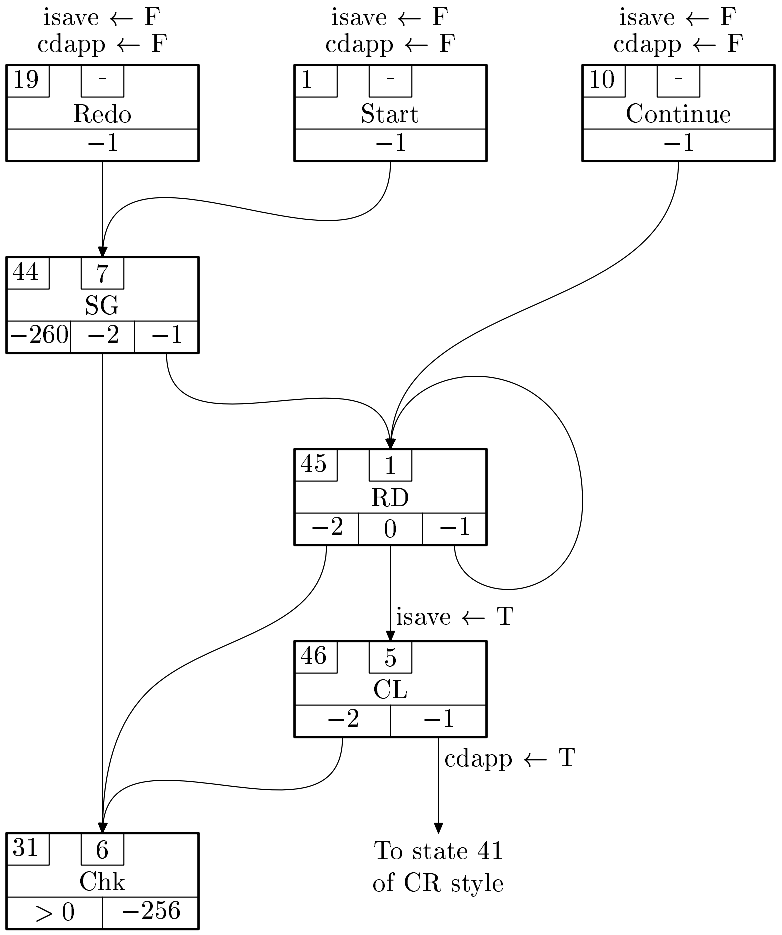


FIGURE C.2: The R* style

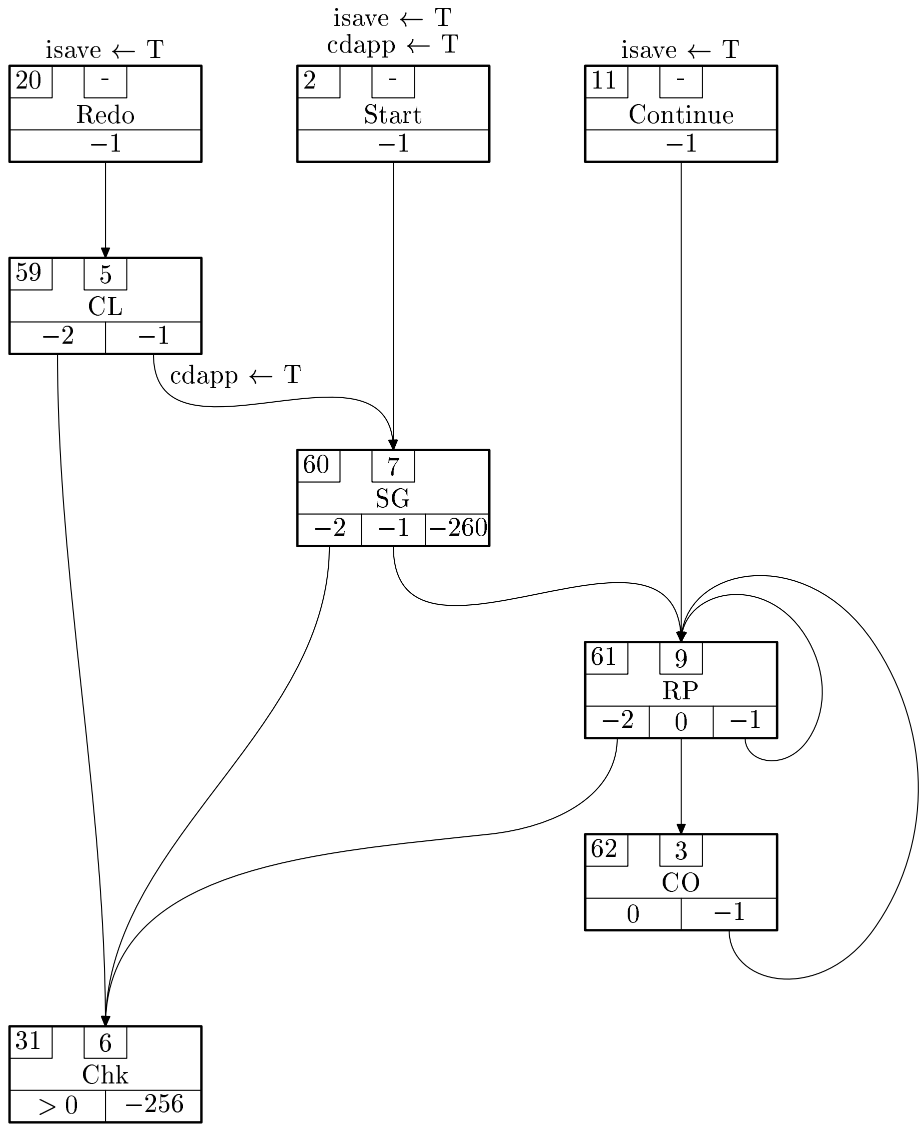


Figure C.3: The Cr style

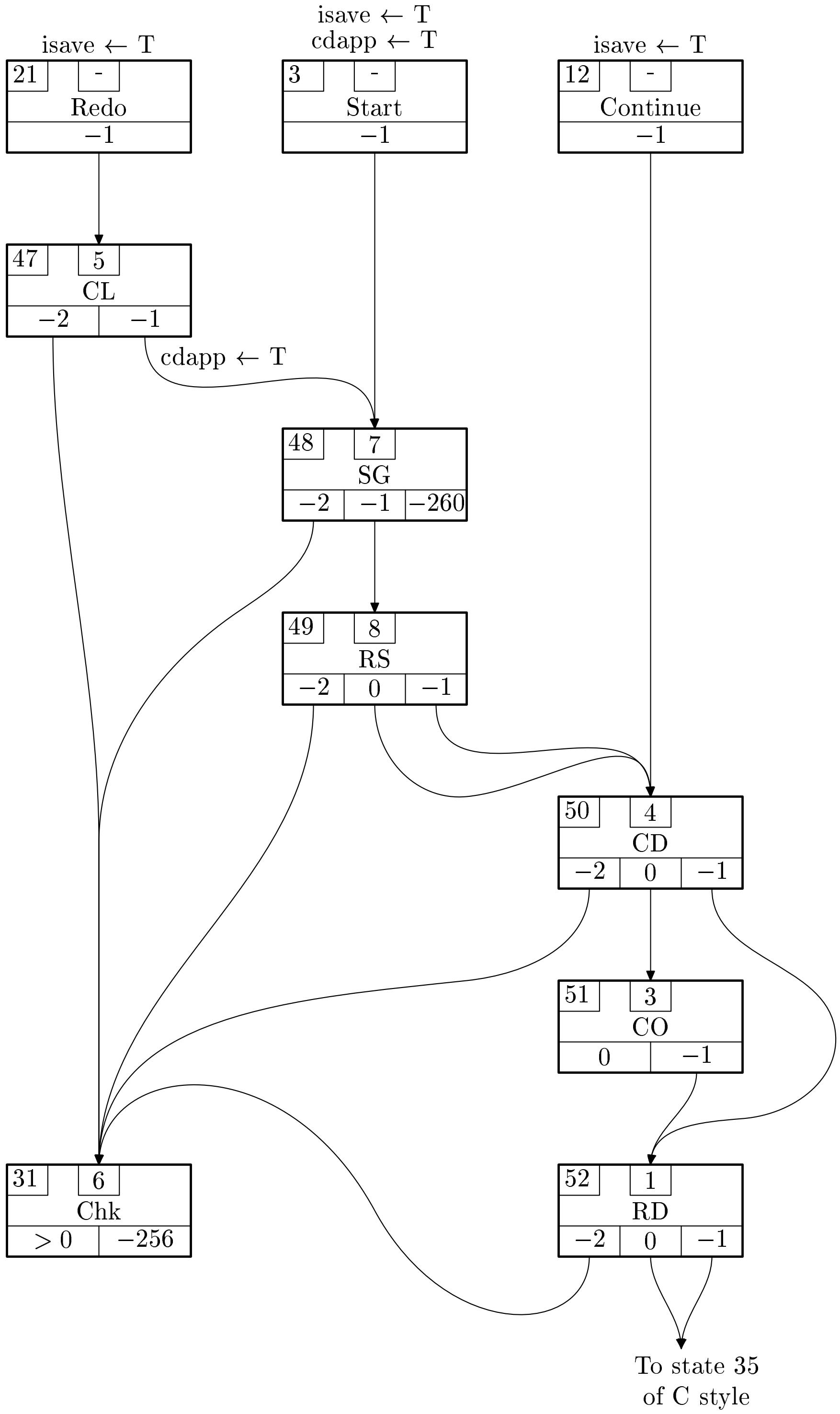


Figure C.4: The C style

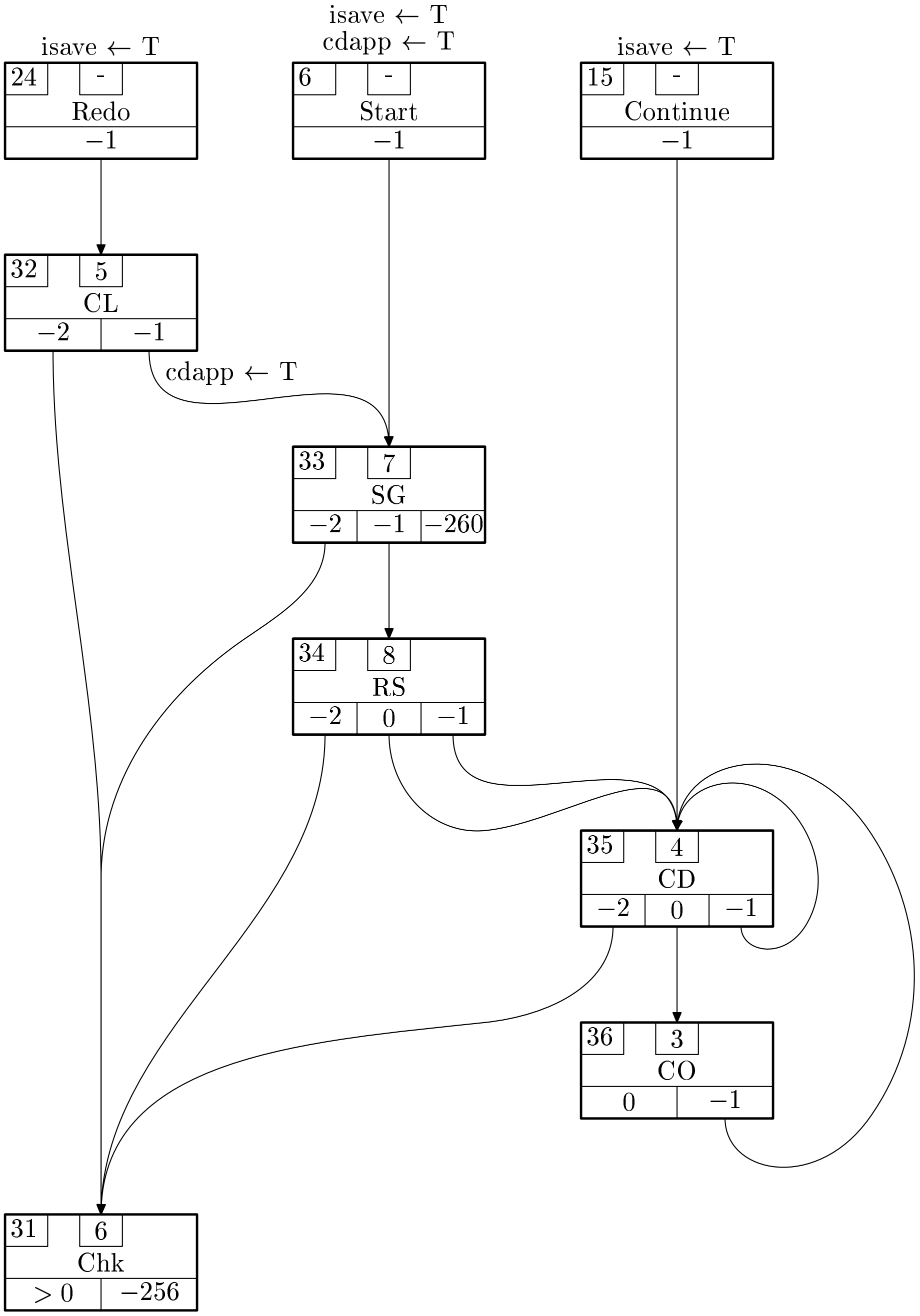


Figure C.5: The Rc style

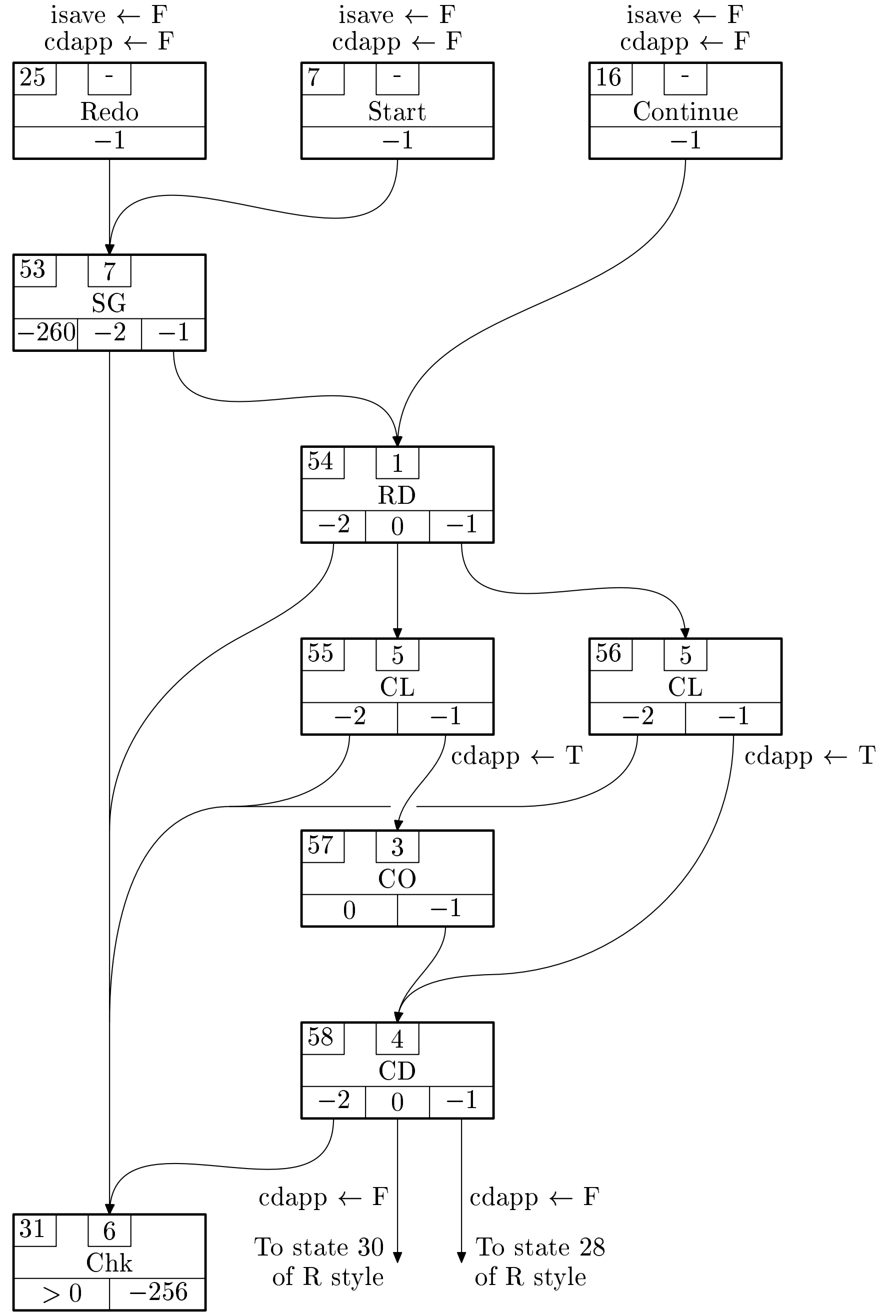


FIGURE C.6: The R style

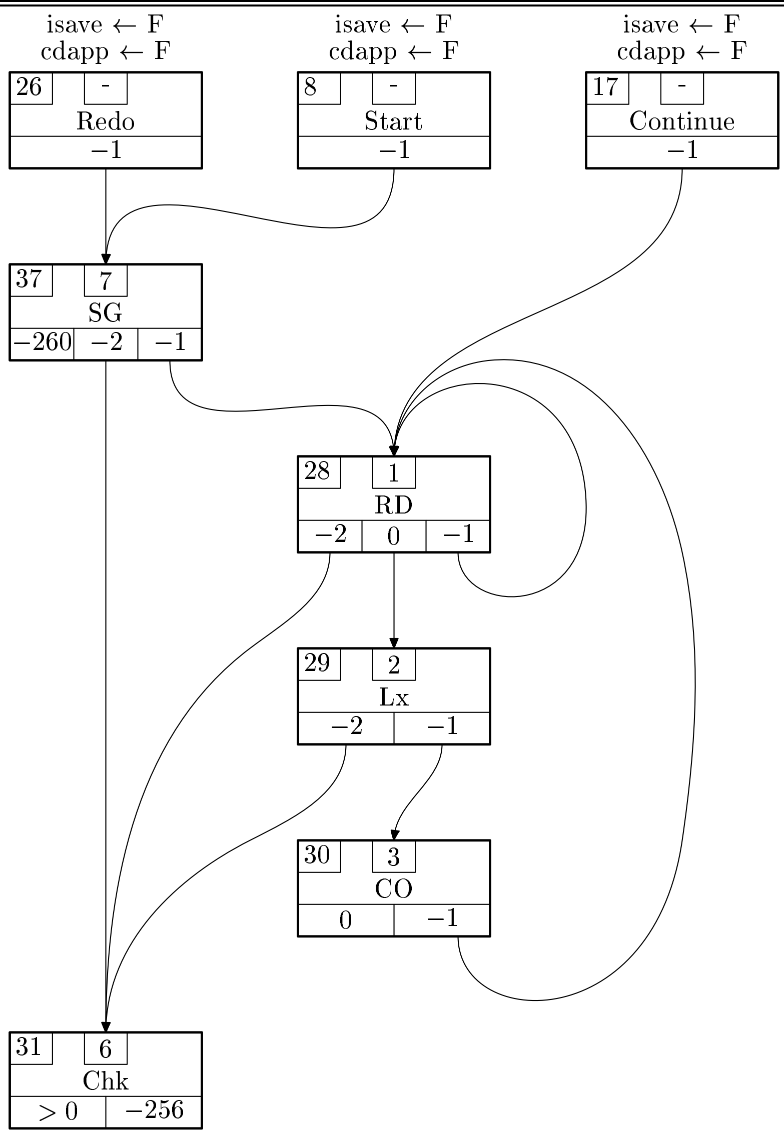
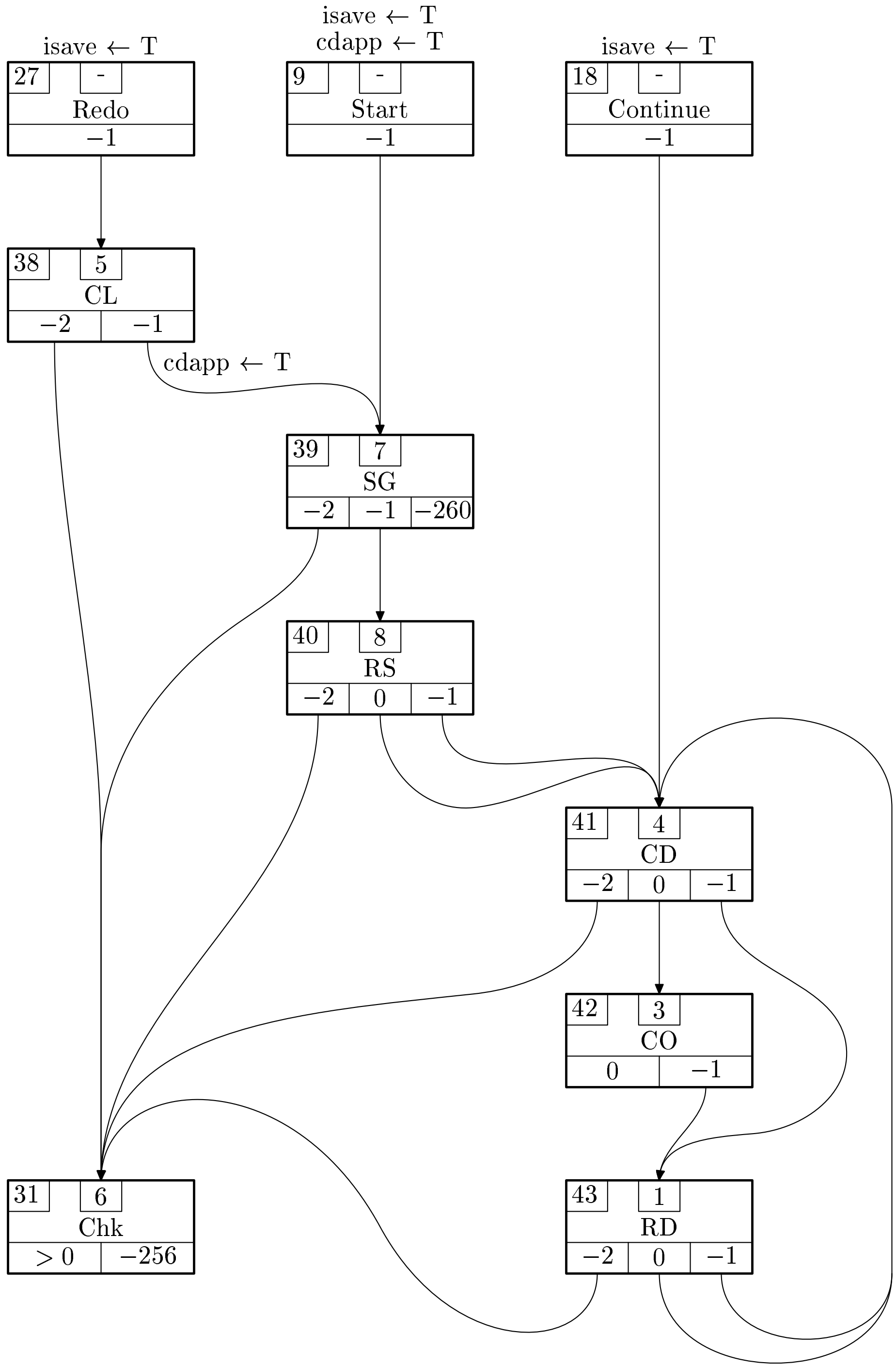


Figure C.7: The CR style



Appendix D

Abbreviations

This appendix lists: the abbreviations and acronyms we use; the technical terms we use; the various terms used in describing ACE, and in explicating its internals; any terms specific to PACE or PEACE which are used. Note that this list includes both terms used in this manual and terms commonly used in the source code.

ACE	advanced coset enumerator
aka	also known as
alive	an active (non-pending/dead) coset
ANSI	American national standards institute
arg	argument
asap	as soon as possible
ave	average
barrier	sync point at which all threads wait until all are ready
`beg(in)`	starts an enumeration ab initio
bn	between
BSD	Berkeley standard distribution
C	the best programming language, *ever*
CC	coinc coset processed (enumeration message/phase)
CD	coset table definition (enumeration message/phase)
cds	complete definition sequence
`check`	synonym for `redo`
Chk	result checking (enumeration message/phase)
CL	coset table based lookahead (enumeration message/phase)
cmd	command
CO	table compaction (enumeration message/phase)
coinc	coincidence. Primary coinc – occurs during defns/scans. Secondary coinc – consequence of a primary one
col	column
concurrent	potentially at the same time, or virtual parallelism
`cont(inue)`	continues the current enumeration
cos	coset
C(PU)	aggregated CPU time for an enumeration
CPU	central processor unit
CRG	the PACE style – coset table, relator tables & gap filling
CT	coset table

DD	serial deduction stack processing (enumeration message/phase)
dead	a fully processed coinc coset
dedn	deduction. Formally – a deduction made during relator scanning. Loosely – any (new/altered) table entry which is stacked
defn	definition
defn seq	definition sequence
DG	serial gap-filling (enumeration message/phase)
`DOSTK`	dedn processing macro, calls appropriate handler
DS	definition sequence
`dtime`	total elapsed time in `DOSTK` macro (part of stats)
DTT	special debug/test/trace code
edp	essentially different position(s)
eg	exempli gratio, for example
elt	element
`end`	synonym for `begin` (don't blame me!)
EOF	end-of-file
EOL	end-of-line
Err	error (enumeration message/phase)
etc	et cetera
F	FALSE
G	the group
gen	generator, either of grp or of subgrp
GNU	GNU's not Unix – quality 'freeware'
$g(p)$	growth function of T with p
grp	group
H	the subgroup
HD	heuristic definition (enumeration message/phase)
ie	id est, that is
inc(l)	include/including
inc(r)	increase/increasing
inv	inverse
invol(n)	involution
I/O	input/output
IP, i/p	input
item	PWs are sequences of items
KISS	keep it simple, stupid
`(kn)h`	coset table rows <`knh` are guaranteed to be complete
`(kn)r`	coset table rows <`knr` are guaranteed to scan at all relators
LC	lower-case
len	length
lst	list
LWP	lightweight process – sorta like a thread, but not quite

`m`, M	MaxCos, the maximum number of cosets active
mode	start, continue or redo an enumeration
mutex	POSIX mutual exclusion lock
n	the number of slaves/threads (i.e., the argument of `beg`)
n/a	not applicable
`n(extdf)`	number of next coset to be defined
`nproc`	global variable containing value of n
NW	non-whitespace (ie, not space, tab, or (maybe) newline
OP, o/p	output
OS	operating system
p	the dedn stack batching factor (i.e., the argument of `pf`)
PACE	parallel ACE
PAR	the parallelisable portion of the running time
para	paragraph
parallel	actually at the same time, or real parallelism
parallel	a PACE run with $n \neq 0$
parentheses	the "(" & ")" characters
PC	proof certificate
pdl	preferred definition list
PEACE	proof extraction after coset enumeration
pending	a coset on the coinc queue but not yet processed
`pfactor`	global variable containing value of p
pthread	POSIX thread
PD	parallel deduction stack processing (enumeration message/phase)
PG	parallel gap-filling (enumeration message/phase)
pos(n)	position
POSIX	portable operating system interface
PPP	paranoia prevent problems (ie, belts'n'braces)
pri	primary
PT	proof table
ptr	pointer
PW	proof word
RD	relator table definition (enumeration message/phase)
RA	relator application check (enumeration message/phase)
`redo`	redo the current enumeration (keeping the table)
red(n)	reduction
redundant	a dead coset
rel	relator and/or relation
rep	the (current) representative of a coincident coset
RS	relators in subgroup (enumeration message/phase)
sec	secondary
semaphore	sync primitive allowing signalling between threads

seq	sequence
SER	the serial portion of the running time
serial	a PACE run using `beg:0`, or an ACE run
SG	subgroup generator (enumeration message/phase)
SMP	shared memory multiprocessor and/or symmetric multiprocessing
spin-lock	sync via sitting in tight loop until a condition is met
square brackets	the "[" & "]" characters
src	source
stats	statistics (package)
`start`	synonym for `begin`
strategy	the overall enumeration method (ie, HLT, Felsch, Sims:n, etc)
style	which of the state machines is active (ie, R, C, CR, etc)
subgrp	subgroup
SYNC	the master-slave synchronisation overhead time
sync	synchronous, synchronisation
`t`, T	TotCos, the total number of cosets defined
T	TRUE
TAB	tabulate character
TBA	to be announced/advised
thread	an independent execution sequence within a process
tuple	4-element record of significant scan, see the `Dlelt` type (in `al0.h`)
UC	upper-case
UH	update hole count check (enumeration message/phase)
vs	versus
W(ALL)	elapsed, or wall, time for an enumeration
wrd	word
WS	white-space; ie, blanks, tabs, & newlines (maybe)

References

[1] M.J. Beetham. Space saving in coset enumeration. In Michael D. Atkinson, editor, *Computational Group Theory*, pages 19–25. Academic Press, 1984.

[2] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system I: the user language. *Journal of Symbolic Computation*, 24:235–265, 1997.

[3] John J. Cannon, Lucien A. Dimino, George Havas, and Jane M. Watson. Implementation and analysis of the Todd-Coxeter algorithm. *Mathematics of Computation*, 27:463–490, 1973.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[5] L.E. Dickson. *Linear Groups, with an exposition of the Galois field theory.* B.G. Teubner, Leipzig, 1901.

[6] George Havas. Coset enumeration strategies. In Stephen M. Watt, editor, *ISSAC'91 (Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation)*, pages 191–199. ACM Press, 1991.

[7] George Havas and Colin Ramsay. Proving a group trivial made easy: a case study in coset enumeration. *Bulletin of the Australian Mathematical Society*, 62:105–118, 2000.

[8] George Havas and Colin Ramsay. Experiments in coset enumeration. In W.M. Kantor and A. Seress, editors, *Groups and Computation III*, number 8 in Ohio State University Mathematical Research Institute Publications, pages 183–192. Walter de Gruyter, 2001.

[9] J. Leech. Coset enumeration on digital computers. *Proceedings of the Cambridge Philosophical Society*, 59:257–267, 1963.

[10] John Leech. Coset enumeration. In Michael D. Atkinson, editor, *Computational Group Theory*, pages 3–18. Academic Press, 1984.

[11] N.S. Mendelsohn. An algorithmic solution for a word problem in group theory. *Canadian Journal of Mathematics*, 16:509–516, 1964. Corrigendum: *Ibid.* 17:505, 1965.

[12] E.H. Moore. Concerning the abstract groups of order $k!$ and $\frac{1}{2}k!$ holohedrically isomorphic with the symmetric and the alternating substitution-groups on k letters. *Proceedings of the London Mathematical Society (1)*, 28:357–366, 1897.

[13] J. Neubüser. An elementary introduction to coset-table methods in computational group theory. In *Groups – St. Andrews 1981*, London Mathematical Society Lecture Note Series 71, pages 1–45. Cambridge University Press, 1982.

[14] M. Schönert et al. *GAP – Groups, Algorithms and Programming.* Lehrstuhl D für Mathematik, Rheinisch-Westfälische Technische Hochschule, Aachen, 1995.

[15] Charles C. Sims. *Computation with finitely presented groups.* Cambridge University Press, 1994.

[16] J.A. Todd and H.S.M. Coxeter. A practical method for enumerating cosets of finite abstract groups. *Proceedings of the Edinburgh Mathematical Society*, 5:26–34, 1936.

[17] J.N. Ward. A note on the Todd-Coxeter algorithm. In R.A. Bryce, J. Cossey, and M.F. Newman, editors, *Group Theory (Canberra, 1975)*, number 573 in Lecture Notes in Mathematics, pages 126–129. Springer-Verlag, 1977.