NAME

Grail – finite-state machines and regular expression software

Grail is a collection of programs for processing finite-state machines and regular expressions. At the user level, Grail consists of a set of filters that manipulate machines and expressions. Machines can be minimized, made deterministic, renumbered, reversed, executed (on some target string), enumerated, completed, complemented, reduced to reachable sets, and converted to regular expressions. Regular expressions can be converted to finite-state machines and have their parenthesization minimized. There are also a set of predicate filters that test for conditions such as determinism, completeness, isomorphism, and universality. The use of these filters is described in the User’s Guide to Grail and in the associated man pages.

Grail defines both conventional and extended finite-state machines. Extended machines permit regular expressions as instruction labels, whereas conventional machines permit only single symbols as instruction labels. For both types of machines, Grail permits multiple start and final states.

Grail is based on a C++ class library that can be called directly from a C++ program. The use of this class library is described in the Programmer’s Guide to Grail.

AUTHORS

Darrell Raymond and Derick Wood

SEE ALSO

fm(5), xfm(5), re(5), fmcomp(1), fmcat(1), fmcross(1), fmenum(1), fmexec(1), fmmin(1), fmminrev(1), fmplus(1), fmreach(1), fmrenum(1), fmreverse(1), fmstar(1), fntore(1), fnuminion(1), iscomp(1), isdeterm(1), isempty(1), isomorph(1), isuniv(1), fmdeterm(1), recat(1), remin(1), restar(1), retofm(1), reunion(1), xfmcat(1), xfmplus(1), xfmreach(1), xfmreverse(1), xfmstar(1), xfntore(1), xfnuminion(1)
NAME
fmcat – catenate two machines

SYNOPSIS
fmcat fm1 fm2

fmcat fm2 <fm1

DESCRIPTION
fmcat computes the catenation of fm1 and fm2, writing the result on the standard output. fm1 and fm2 need not be distinct. fmcat does not introduce empty-string instructions. It catenates the machines by connecting the final states of fm1 to the targets of start states in fm2, and appending any other instructions. Before catenation, the states in fm2 are renumbered so there are no collisions with states in fm1.

fm1 and fm2 must conform to the Grail format for machines.

EXAMPLES
% cat dfm1
(START) | - 0
0 a 1
1 b 2
2 - | (FINAL)

% fmcat dfm1 dfm1
(START) | - 0
0 a 1
1 b 2
2 a 4
4 b 5
5 - | (FINAL)

% cat nfm2
(START) | - 1
1 a 2
1 a 3
1 a 4
2 - | (FINAL)
3 - | (FINAL)
4 - | (FINAL)
% fmcat nfm2 dfm1
(START)  |  1
 1 a 2
 1 a 3
 1 a 4
 2 a 6
 3 a 6
 4 a 6
 6 b 7
 7 - | (FINAL)

**AUTHORS**
Darrell Raymond and Derick Wood

**SEE ALSO**
fm(5)
NAME
fmcment – compute the complement of a machine

SYNOPSIS
fmcment fm

fmcment <fm

DESCRIPTION
fmcment computes the complement of fm and writes the result on the standard output. fmcment performs subset construction if the machine is not deterministic, and completion if the machine is incomplete.

fm must conform to the Grail format for machines.

The complement of a machine accepts any string not accepted by the original machine. Complement is defined in terms of the underlying alphabet of the machine. Since Grail machines do not contain a separate specification of their underlying alphabet, fmcment assumes that the alphabet used in the input machine is the underlying alphabet. Thus, fmcment computes the complement only with respect to the symbols that appear in the original machine. In order to compute complement with respect to an alphabet containing symbols that are not in the original machine, it is necessary to add instructions from a start state to a new non-final state, one instruction for each missing symbol. The new state should be the source of no instructions.

EXAMPLES
% cat dfm1
(START) |- 0
0 a 1
1 b 2
2 - | (FINAL)

% fmcment dfm1
0 a 1
1 b 2
0 b 3
1 a 3
2 a 3
2 b 3
3 a 3
3 b 3
(START) |- 0
% cat nfm2
(START)  |- 1
1 a 2
1 a 3
1 a 4
2 -  (FINAL)
3 -  (FINAL)
4 -  (FINAL)

% fmcment <nfm2
0 a 1
1 a 2
2 a 2
(START)  |- 0
0 -  (FINAL)
2 -  (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5)
NAME
fmcomp – compute the completion of a machine

SYNOPSIS
fmcomp fm

fmcomp <fm

DESCRIPTION
fmcomp computes the completion of fm and writes the result on the standard output.

fm must conform to the Grail format for machines.

A complete machine is one in which every state has a instruction on every symbol in the alphabet. fmcomp completes its input by creating a new ‘sink’ state that is used as the target of any missing instructions in the input machine.

EXAMPLES
% cat dfm1
(START) |- 0
0 a 1
1 b 2
2 -| (FINAL)

% fmcomp dfm1
(START) |- 0
0 a 1
1 b 2
2 -| (FINAL)
0 b 3
1 a 3
2 a 3
2 b 3
3 a 3
3 b 3

% cat nfm2
(START) |- 1
1 a 2
1 a 3
1 a 4
2 -| (FINAL)
AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5)
fmcross – compute the cross product of two machines

fmcross fm1 fm2

fmcross fm2 <fm1

DESCRIPTION

fmcross computes the cross product of the machines fm1 and fm2, writing the result machine on the standard output. Both machines may be specified on the command line, or one may be read from standard input. fm2 can, if desired, be the same file as fm1.

The result is not guaranteed to have a final state unless fm2 is the same as fm1. Furthermore, the generated machine is not guaranteed to be complete, connected, minimal, or deterministic.

If two machines are not specified, fmcross returns 0. fm1 and fm2 must conform to the Grail format for machines.

The cross product contains a instruction of the form:

\[(x_{fm1}, label, (y_{fa1}, y_{fa2}))\]

for each pair of instructions in the input machines of the form

\[(x_{fm1}, label, y_{fa1}) \in fa1\]
\[(x_{fm2}, label, y_{fa2}) \in fa2\]

The state numbers in the output machines are computed from the input state numbers as follows:

\[s_0 = s_{fm1} + ((\text{max}+1)*s_{fm2})\]

where \(s_0\) is the output state number, \(s_{fm2}\) is the state number of fm2, and max is the maximum state number of fm1. Since the output state numbers have a unique factorization in terms of input state numbers, it is possible to determine from the output state which pair of input states it represents.

Computing the cross product of two finite-state machines generates their intersection; if the input machines are equivalent, then the result is the same as the input. Computing the cross product of a nondeterministic machine with itself produces a result that accepts the same language, but is substantially larger. Recursive application of cross product results in an exponential growth in the size of the machine. Thus one can generate large nondeterministic machines with a known language; this may be useful for testing other filters.
fmcross requires space proportional to its result. Recursive cross product of even the smallest nondeterministic machines more than four or five times will consume tens of megabytes of memory.

EXAMPLES
This example computes the cross product of a simple nfm with itself:

```
% cat nfm
(START) | 0
0 a 1
0 a 2
1 -| (FINAL)
2 -| (FINAL)

% fmcross nfm nfm
0 a 4
0 a 7
0 a 5
0 a 8
(START) | 0
4 -| (FINAL)
7 -| (FINAL)
5 -| (FINAL)
8 -| (FINAL)
```

This example computes the cross product of two fms which have the property that $L_1 \subseteq L_2$:

```
% cat dfm1
(START) | 0
0 a 1
1 b 2
2 c 3
3 -| (FINAL)

% cat dfm2
(START) | 0
0 a 0
0 b 1
1 c 1
1 -| (FINAL)
```
% fmcross dfm1 dfm2
0 a 1
1 b 6
6 c 7
(START) | - 0
7 - (FINAL)

This example shows the exponential increase in the size of cross product results, using wc to compute the size of the machine file):

$ for i in 1 2 3 4
  > do
  >   fmcross nfm nfm >tmp
  >   mv tmp nfm
  >   wc nfm
  > done
  
  9   27    97 nfm
  33   99   381 nfm
  513 1539  6925 nfm
 131073 393219 2293773 nfm
$

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5)
NAME
fmdeterm – make a machine deterministic

SYNOPSIS
fmdeterm fm

fmdeterm <fm

DESCRIPTION
fmdeterm computes a deterministic machine from fm, using the subset construction method. In a small number of cases, this will cause an exponential increase in the size of the machine.

fm must conform to the Grail format for machines.

EXAMPLES

% cat nfm1
(START) |- 1
 1 a 2
 1 a 3
 2 b 2
 3 b 3
 2 c 4
 3 c 5
 4 d 4
 5 d 5
 4 |- (FINAL)
 5 |- (FINAL)

% fmdeterm nfm1
(START) |- 0
 0 a 1
 1 b 1
 1 c 2
 2 d 2
 2 |- (FINAL)

% cat nfm2
(START) |- 1
 1 a 2
 1 a 3
 1 a 4
% fmdeterm <nfm2
(START) | - 0
  0 a 1
  1 - | (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), isdeterm(1)
NAME
fmenum – enumerate the language of a machine

SYNOPSIS
fmenum fa [num]
fmenum [num] <fm

DESCRIPTION
fmenum enumerates the language of fm and writes the strings on its standard output. It produces 100 strings (or num strings, if num is specified) that belong to the language of fm. fmenum can enumerate the language of both deterministic and nondeterministic machines. fmenum produces strings in order of their length, shortest first; within the same length, they are lexicographically ordered. fm must conform to the Grail format for machines.

EXAMPLES
% cat nfm1
(START) |- 1
  1 a 2
  1 a 3
  2 b 2
  3 b 3
  2 c 4
  3 c 5
  4 d 4
  5 d 5
  4 - | (FINAL)
  5 - | (FINAL)

% fmenum 10 nfm1
ac
abc
acd
abbc
abcd
acdd
abbbcd
abbcdd
acddd

January 1994 Grail 1
% fmenum foobar <nfml
fmenum: enumeration value foobar invalid

% fmenum 5 <nfml
ac
abc
acd
abbc
abcd

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmexec(1)
NAME
fmexec – execute a machine on an input string

SYNOPSIS
fmexec [-d] fa string

    fmexec [-d] string <fm

DESCRIPTION
fmexec tests string for acceptance in the language of the machine fm. If string is
accepted, fmexec returns 1 and writes accepted on its standard error; otherwise
it returns 0 and writes not accepted on its standard error. fmexec can execute
both deterministic and nondeterministic machines.

The -d option causes fmexec to print each instruction that it executes for each
character of string that is processed. In the case of nondeterministic machines,
fmexec will print the set of instructions that are executed for each character of
string.

fm must conform to the Grail format for machines. string should probably be
protected by double quotes.

EXAMPLES

% cat nfm1
(START) |- 1
  1 a 2
  1 a 3
  2 b 2
  3 b 3
  2 c 4
  3 c 5
  4 d 4
  5 d 5
  4 - | (FINAL)
5 - | (FINAL)

% fmexec nfm1 "abc"
accepted

% fmexec nfm1 "abbbbbbbbbbcdddddddddd" accepted

% fmexec nfm1 "x"
not accepted

% fmexec -d "abbcdd" <nfm1
on a take instructions
 1 a 2
 1 a 3
on b take instructions
 2 b 2
 3 b 3
on b take instructions
 2 b 2
 3 b 3
on c take instructions
 2 c 4
 3 c 5
on d take instructions
 4 d 4
 5 d 5
terminate on final states 4 5

accepted

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmenum(1)
NAME
fmmin – compute the minimal machine

SYNOPSIS
fmmin fm

fmmin <fm

DESCRIPTION
fmmin computes the minimal machine that accepts the same language as fm, and writes the result on the standard output. fmmin returns 0 if the input machine is non-deterministic. The machine can be made deterministic by first filtering it with fmdeterm. fmmin uses Hopcroft’s partition algorithm. It does not remove unreachable states.

fm must conform to the Grail format for machines.

EXAMPLES
% cat dfm
(START) | - 0
0 a 1
0 b 2
1 c 1
2 c 2
1 d 3
2 d 4
3 - | (FINAL)
4 - | (FINAL)

% fmmin dfm
(START) | - 2
2 a 1
2 b 1
1 c 1
1 d 0
0 - | (FINAL)

% cat nfm2
(START) | - 1
1 a 2
1 a 3
1 a 4
% cat nfm2 | fmdeterm | fmmin
(START) | - 1
1 a 0
0 - | (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmminrev(1), fmdeterm(1)
NAME
fmminrev – compute the minimal machine

SYNOPSIS
fmminrev fm

fmminrev <fm

DESCRIPTION
fmminrev computes the minimal machine that accepts the same language as fm, and writes the result on the standard output. fmminrev returns 0 if the input machine is nondeterministic. The machine can be made deterministic by filtering it with fmdeterm.

fmminrev computes the minimal machine by reversing, performing subset construction (that is, by applying fmdeterm), reversing again, and performing subset construction a final time). The result is guaranteed to be deterministic.

Machines can also be minimized by fmmin fa, which uses Hopcroft’s partition method. fmmin and fmminrev should produce isomorphic results (that is, identical up to state renumbering).

fm must conform to the Grail format for machines.

EXAMPLES
% cat dfm
(START) |- 0
  0 a 1
  0 b 2
  1 c 1
  2 c 2
  1 d 3
  2 d 4
  3 -| (FINAL)
  4 -| (FINAL)

% fmminrev <dfm
(START) |- 0
  0 a 1
  0 b 1
  1 d 2
  1 c 1
  2 -| (FINAL)
% cat nfm2
(START) |− 1
1 a 2
1 a 3
1 a 4
2 − | (FINAL)
3 − | (FINAL)
4 − | (FINAL)

% cat nfm2 | fmdeterm | fmminrev
(START) |− 0
0 a 1
1 − | (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmmin(1), fmreverse(1), fmdeterm(1), ismorph(1)
fmplus \((1)\) User Commands fmplus \((1)\)

NAME
fmplus – compute ‘+’ of machine

SYNOPSIS
fmplus fm

fmplus <fm

DESCRIPTION
fmplus computes the ‘+’ of fm; that is, the machine accepting one or more occurrences of words accepted by fm. The result is written on standard output.

fmplus can be applied to either deterministic or nondeterministic machines. The result is guaranteed to be nondeterministic.

fm must conform to the Grail format for machines.

fmplus computes ‘+’ by making all instructions to final states also go to start states. The result has no empty-string instructions.

EXAMPLES

```
% cat dfm1
(START) |- 0
  0 a 1
  1 b 2
  2 -| (FINAL)

% fmplus dfm1
(START) |- 0
  0 a 1
  1 b 2
  2 -| (FINAL)
  1 b 0

% cat nfm2
(START) |- 1
  1 a 2
  1 a 3
  1 a 4
  2 -| (FINAL)
  3 -| (FINAL)
  4 -| (FINAL)

% fmplus <nfm2
```

January 1994 Grail 1
fmplus (1)  User Commands  fmplus (1)

(START)  |  1
  1 a 2
  1 a 3
  1 a 4
  2 -  (FINAL)
  3 -  (FINAL)
  4 -  (FINAL)
  1 a 1

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5)

January 1994  Grail  2
NAME
fmreach – compute the reachable subset of a machine

SYNOPSIS
fmreach fm

fmreach <fm

DESCRIPTION
fmreach finds all reachable states of fm and writes on its standard output only
instructions involving those states.

fm must conform to the Grail format for machines.

EXAMPLES
% cat dfm4
(START) |- 0
0 a 1
0 g 0
0 b 4
1 c 2
2 d 3
3 -| (FINAL)
4 e 5
5 f 6
6 -| (FINAL)

% fmreach <dfm4
(START) |- 0
0 a 1
0 g 0
0 b 4
1 c 2
2 d 3
3 -| (FINAL)
4 e 5
5 f 6
6 -| (FINAL)

% cat dfm6
(START) |- 3
3 a 4
% fmreach dfm6
(START) | - 3
3 a 4
4 b 5
5 - | (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), xfmreach(1)
NAME
fmrenum – renumber a machine

SYNOPSIS
fmrenum fm

fmrenum <fm

DESCRIPTION
fmrenum renumbers the states in fm according to a canonical numbering; breadth-first and lexicographically on the instruction labels. The renumbered machine is placed on standard output.

If isomorphic machines are canonically renumbered, they are identical.

fmrenum returns 0 and writes a message on standard error if fm is nondeterministic. A machine can be made deterministic by filtering it with fmdeterm. fm must conform to the Grail format for machines.

EXAMPLES

% cat dfm2
(START) │ - 3
  3 a 4
  4 b 5
  5 - │ (FINAL)

% fmrenum dfm2
(START) │ - 0
  0 a 1
  1 b 2
  2 - │ (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmdeterm(1)

January 1994 Grail 1
NAME
fmreverse – reverse a machine

SYNOPSIS
fmreverse fm

DESCRIPTION
fmreverse reverses the direction of all instructions in fm and writes the result on standard output. All start states become final states and vice versa. The input need not be deterministic. The output will be nondeterministic if fm contains more than one final state (since a deterministic machine can have only one start state).

fm must conform to the Grail format for machines.

EXAMPLES

% cat dfm5
(START) |-- 0
 0 a 1
 1 c 2
 2 e 3
 3 -- (FINAL)
 1 b 0
 2 d 0

% fmreverse dfm5
0 -- (FINAL)
 1 a 0
 2 c 1
 3 e 2
(START) |-- 3
 0 b 1
 0 d 2

% cat nf
(START) |-- 1
 1 a 2
 1 a 3
 1 a 4
 2 -- (FINAL)
fmreverse(1) User Commands fmreverse(1)

3 - | (FINAL)
4 - | (FINAL)

% fmreverse <nfm2
1 - | (FINAL)
2 a 1
3 a 1
4 a 1

(START) | - 2
(START) | - 3
(START) | - 4

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5)
NAME
fmstar – compute ‘*’ of a machine

SYNOPSIS
fmstar fm

fmstar <fm

DESCRIPTION
fmstar computes ‘*’ (also known as Kleene closure) of fm and writes the result on standard output. The input need not be deterministic.

fm must conform to the Grail format for machines.

fmstar introduces no empty-string instructions. It first computes the ‘+’ of fm, then it clones the start state and makes it a final state.

EXAMPLES
% cat dfm5
(START) |- 0
  0 a 1
  1 c 2
  2 e 3
  3 |- (FINAL)
  1 b 0
  2 d 0

% fmstar dfm5
  0 a 1
  1 c 2
  2 e 3
  3 |- (FINAL)
  1 b 0
  2 d 0
  2 e 0
  4 a 1
  4 |- (FINAL)
(START) |- 4
AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fmt(5), fmplus(1)
NAME
fmtore – convert a machine to a regular expression

SYNOPSIS
fmtore fm
fmtore <fm

DESCRIPTION
fmtore computes a regular expression that accepts the same language as fm, and
writes the result on standard output. The input need not be deterministic.
fmtore uses the state elimination method for producing the regular expression.
fm must conform to the Grail format for machines.

EXAMPLES
% cat dfm5
(START) |- 0
0 a 1
1 c 2
2 e 3
3 - | (FINAL)
1 b 0
2 d 0

% fmtore <dfm5
a(ba)*c(da(ba)*c)*e

% cat nfm1
(START) |- 1
1 a 2
1 a 3
2 b 2
3 b 3
2 c 4
3 c 5
4 d 4
5 d 5
4 - | (FINAL)
5 - | (FINAL)

% fmtore nfm1
ab*cd*

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), re(5), retofm(1)
NAME
fmunion – compute the union of two machines

SYNOPSIS
fmunion fm1 fm2

fmunion fm2 <fm1

DESCRIPTION
fmunion computes the union of fm1 and fm2. This is done by renumbering the states of fm2 and then appending its instructions to those of fm1. The input need not be deterministic.

fm1 and fm2 must conform to the Grail format for machines.

EXAMPLES
% cat dfm1
(START) |- 0
  0 a 1
  1 b 2
  2 -| (FINAL)

% cat dfm3
(START) |- 0
  0 a 1
  0 b 4
  1 c 2
  2 d 3
  3 -| (FINAL)
  4 e 5
  5 f 6
  6 -| (FINAL)

% fmunion dfm1 dfm3
(START) |- 0
  0 a 1
  1 b 2
  2 -| (FINAL)
  (START) |- 3
  3 a 4
  3 b 7
  4 c 5

January 1994 Grail 1
AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmrenum(1)
NAME
iscomp – test for completeness

SYNOPSIS

iscomp fm

iscomp <fm

DESCRIPTION

iscomp tests fm for completeness (that every state has a instruction with every instruction label). The input alphabet is considered to be the set of labels present in the input machine. iscomp returns 1 and writes complete on standard output if fm is complete; otherwise, it returns 0 and writes incomplete.

An incomplete machine can be made complete with fmcomp.

fm must conform to the Grail format for machines.

EXAMPLES

% cat dfm3
(START) | - 0
0 a 1
0 b 4
1 c 2
2 d 3
3 - | (FINAL)
4 e 5
5 f 6
6 - | (FINAL)

% iscomp dfm3
incomplete

% fmcomp dfm3 | iscomp
complete

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmcomp(1)
NAME
isdeterm – test machine for ‘determinism’

SYNOPSIS
isdeterm fm
isdeterm <fm

DESCRIPTION
isdeterm checks if fm is deterministic. isdeterm returns 1 and writes deterministic on standard output if the input fm is deterministic; otherwise, it returns 0 and writes nondeterministic.

A nondeterministic machine can be made deterministic with fmdeterm.

fm must conform to the Grail format for machines.

EXAMPLES
% cat nfm1
(START) │ − 1
1 a 2
1 a 3
2 b 2
3 b 3
2 c 4
3 c 5
4 d 4
5 d 5
4 − | (FINAL)
5 − | (FINAL)

% isdeterm nfm1
nondeterministic

% fmdeterm nfm1 | isdeterm
deterministic

AUTHORS
Darrell Raymond and Derick Wood

January 1994 Grail 1
NAME
isempty – test re for containment of empty set

SYNOPSIS
isempty re
isempty <re

DESCRIPTION
isempty tests re to see if it is the empty set. isempty returns 1 and writes is empty set on standard output if re is the empty set; it returns 0 and writes is not empty set otherwise.

re must conform to the Grail format for regular expressions.

EXAMPLES
% cat re1
{}

% isempty re1
is empty set

% cat re2
"

% isempty re2
is not empty set

% cat re3
(a+b)*(abc)

% isempty re3
is not empty set

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
re(5), isnull(1)
NAME
isnull – test re for equivalence to empty string

SYNOPSIS
isnull re

isnull <re

DESCRIPTION
isnull tests re to see if it is the empty string. isnull returns 1 and writes is empty string on standard Output if re is the empty string; it returns 0 and writes is not empty string otherwise.

re must conform to the Grail format for regular expressions.

EXAMPLES
% cat re1
{}
% isnull re1
is not empty string

% cat re2
"
% isnull re2
is empty string

% cat re3
(a+b)*(abc)
% isnull re3
is not empty string

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
re(5), isempty(1)
NAME
isomorph – test two machines for isomorphism

SYNOPSIS
isomorph fm1 fm2

isomorph fm2 <fm1

DESCRIPTION
isomorph tests fm1 and fm2 for isomorphism. isomorph returns 1 and writes isomorphic on standard output if the two machines are isomorphic, and returns 0 and writes nonisomorphic otherwise.

If two machines are not input, isomorph writes a diagnostic on standard error and returns 0. If either machine is not deterministic, isomorph returns -1 and writes a diagnostic on its standard error. A machine can be made deterministic by filtering it with fmdetermin.

Two machines are isomorphic if they are equivalent up to renumbering. Isomorphism is checked by applying canonical numbering to each machine and then testing for identity.

fm1 and fm2 must conform to the Grail format for machines.

EXAMPLES

% cat dfm4
(START) | 0
  0 a 1
  0 g 0
  0 b 4
  1 c 2
  2 d 3
  3 | (FINAL)
  4 e 5
  5 f 6
  6 | (FINAL)

% isomorph dfm4 dfm4
isomorphic

% cat dfm1
(START) | 0
isomorph (1)

User Commands

isomorph (1)

0 a 1
1 b 2
2 -| (FINAL)

% cat dfm2
(START) | - 3
3 a 4
4 b 5
5 -| (FINAL)

% isomorph dfm1 dfm2
isomorphic

% isomorph dfm1 dfm4
non-isomorphic

% isomorph dfm1 nfml
second machine is not deterministic

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
fm(5), fmdeterm(1), isdeterm(1)
NAME
isuniv – test machine for universality

SYNOPSIS
isuniv fa

isuniv <fa

DESCRIPTION
isuniv tests if fa is universal—that is, complete and all reachable states are also
final states. isuniv returns 1 and writes universal on standard output if the input
fa is universal; it returns 0 and writes nonuniversal otherwise.

fa must conform to the Grail format for machines.

EXAMPLES
% cat dfm6
(START) | - 0
  0 a 1
  0 b 2
  0 -| (FINAL)
  1 b 2
  1 a 0
  2 a 1
  2 b 2
  1 -| (FINAL)
  2 -| (FINAL)

% isuniv dfm6
universal

% cat dfm1
(START) | - 0
  0 a 1
  1 b 2
  2 -| (FINAL)

% isuniv dfm1
nonuniversal

January 1994 Grail 1
AUTHORS
   Darrell Raymond and Derick Wood

SEE ALSO
   fm(5), fmcomp(1)
NAME
recat – catenate two regular expressions

SYNOPSIS
recat re1 re2
recat re2 <re1

DESCRIPTION
recat catenates re1 with re2, and writes the result on standard output.

re1 and re2 must conform to the Grail format for regular expressions.

EXAMPLES
% cat re1
{}
%
% cat re2
**
%
% cat re3
(a+b)* (abc)
%
% recat re1 re3
{}
%
% recat re2 re3
(a+b)*abc
%
% recat re3 re3
(a+b)*abc (a+b)*abc

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
re(5)
NAME
remin – produce minimal parenthesization of a regular expression

SYNOPSIS
remin re

remin <re

DESCRIPTION
remin produces the minimal parenthesization of re, and applies some simple heuristics for minimizing the expression (removes subexpressions that are catenated with the empty set, removes the empty string from catenations, and removes redundant subexpressions in unions).

Any other Grail filter for regular expressions will remove superfluous parenthesis, simply by virtue of reading and writing an expression.

re must conform to the Grail format for regular expressions.

EXAMPLES
% cat re1
{}

% remin <re1
{}

% cat re2

% remin re2

% cat re3
(a+b)*abc

% remin re3
(a+b)*abc

% cat re4
(((a)+(b))*

% remin re4
(a+b)*

January 1994 Grail 1
AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
re(5)
NAME
restar – compute ‘∗’ of a regular expression

SYNOPSIS
restar re

restar <re

DESCRIPTION
restar computes the Kleene star of re, and writes the result on standard output.

re must conform to the Grail format for regular expressions.

EXAMPLES
% cat re1
{}

% restar <re1
{}

% cat re2
**

% restar re2
**

% cat re3
(a+b)∗(abc)

% restar <re3
((a+b)∗abc)∗

% cat re4
((((a) + (b)))∗)

% restar re4
(a+b)∗

AUTHORS
Darrell Raymond and Derick Wood

January 1994
Grail

1
NAME
retofm – convert a regular expression to a machine

SYNOPSIS
retofm re

retofm <re

DESCRIPTION
retofm computes a finite-state machine that accepts the same language as re, and writes it on standard output. The result is likely to be nondeterministic.

re must conform to the Grail format for regular expressions.

EXAMPLES
% cat re1
{}

% retofm <re1

% cat re2
**

% retofm <re2
(START) |− 0
0 −| (FINAL)

% cat re3
(a+b)*(abc)

% retofm re3
0 a 1
2 b 3
0 a 0
0 a 2
2 b 0
2 b 2
4 a 1
4 a 0
4 a 2
4 b 3
AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
re(5), fm(5), fmto(1)
NAME
reunion – compute the disjunction of two regular expressions

SYNOPSIS
reunion re1 re2
reunion re2 <re1

DESCRIPTION
reunion computes re1 ‘or’ re2, and writes the result on standard output.
re1 and re2 must conform to the Grail format for regular expressions.

EXAMPLES
% cat re3
(a+b)* (abc)

% cat re2
**

% reunion re3 re2
(a+b)*abc+"

% cat re4
((a+(b))*

% reunion re4 re3
(a+b)*+(a+b)*abc

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
re(5)
NAME
xfmcat – catenate two extended machines

SYNOPSIS
xfmcat xfm1 xfm2

xfmcat xfm2 <xfm1

DESCRIPTION
xfmcat computes the catenation of xfm1 and xfm2, writing the result on the standard output. xfm1 and xfm2 need not be distinct. xfmcat does not introduce empty-string instructions. It catenates the machines by connecting the final states of xfm1 to the targets of start states in xfm2, and appending any other instructions. Before catenation, the states in xfm2 are renumbered so there are no collisions with states in xfm1.

xfm1 and xfm2 must conform to the Grail format for extended machines. Since every conventional machine is also an extended machine, xfmcat can be used to catenate two conventional machines.

EXAMPLES
% cat xfm1
(START) | 0
  0 ab* 1
  1 (c+d)* 2
  2 -| (FINAL)

% xfmcat xfm1 xfa1
(START) | 0
  0 ab* 1
  1 (c+d)* 2
  2 ab* 4
  4 (c+d)* 5
  5 -| (FINAL)

% cat dfm4
(START) | 0
  0 a 1
  0 g 0
  0 b 4
  1 c 2
  2 d 3

January 1994 Grail
3 -| (FINAL)
4 e 5
5 f 6
6 -| (FINAL)

% xfmcat dfm4 xfm1
(START) | - 0
0 a 1
0 g 0
0 b 4
1 c 2
2 d 3
4 e 5
5 f 6
3 ab* 8
6 ab* 8
8 (c+d)* 9
9 -| (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
xfm(5), fmcat(1)
NAME
xfmplus – compute ‘+’ of an extended machine

SYNOPSIS
xfmplus xfm

xfmplus <xfm

DESCRIPTION
xfmplus computes the ‘+’ of xfm; that is, the machine accepting one or more
occurrences of words accepted by xfm. The result is written on standard output.

fmplus computes ‘+’ by making all instructions to final states also go to start
states. The result has no empty-string instructions.

xfm must conform to the Grail format for extended machines. Since every con-
ventional machine is also an extended machine, xfmplus can be used to compute
the ‘+’ of conventional machines.

EXAMPLES

% cat xfa3
(START)   | 0
0 a* 1
0 b* 2
0 c* 3
1 bb* 3
2 cc* 3
3  | (FINAL)

% xfmplus xfa3
(START)   | 0
0 a* 1
0 b* 2
0 c* 3
1 bb* 3
2 cc* 3
3  | (FINAL)
0 c* 0
1 bb* 0
2 cc* 0

% cat dfm4
(START)   | 0

January 1994 Grail 1
AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
xfm(5), fmplus(1)
NAME
xfmreach – compute the reachable subset of an extended machine

SYNOPSIS
xfmreach xfm

xfmreach <xfm

DESCRIPTION
xfmreach finds all reachable states of xfm and writes on its standard output only
instructions involving those states.

xfm must conform to the Grail format for extended machines. Since every con-
ventional machine is an extended machine, xfmreach can also be used to com-
pute reachability for conventional machines.

EXAMPLES
% cat xfm1
(START) |- 0
0 ab* 1
1 (c+d)* 2
2 -| (FINAL)

% xfmreach xfm1
(START) |- 0
0 ab* 1
1 (c+d)* 2
2 -| (FINAL)

% cat dfm6
(START) |- 3
3 a 4
4 b 5
5 -| (FINAL)
1 a 2
2 b 6
6 -| (FINAL)

% xfmreach dfm6
(START) |- 3
3 a 4
4 b 5
AUTHORS
   Darrell Raymond and Derick Wood
SEE ALSO
   xfm(5), fmreach(1)
NAME
xfmreverse – reverse an extended machine

SYNOPSIS
xfmreverse xfm

xfmreverse <xfm

DESCRIPTION
xfmreverse reverses the direction of all instructions in xfm and writes the result on standard output. All start states become final states and vice versa. The output will be non-deterministic if xfm contains more than one final state (a deterministic machine can have only one start state).

xfm must conform to the Grail format for extended machines. Since every conventional machine is also an extended machine, xfmreverse can also be used to reverse conventional machines.

EXAMPLES

% cat xfm2

(START) |- 0
0 " " 1
1 a 2
0 b 2
2 - | (FINAL)

% xfmreverse xfm2

0 - | (FINAL)
1 " " 0
2 a 1
2 b 0
(START) |- 2

% cat dfm5

(START) |- 0
0 a 1
1 c 2
2 e 3
3 - | (FINAL)
1 b 0
2 d 0
xfmreverse(1)  User Commands  xfmreverse(1)

% xfmreverse <dfm5
0 - | (FINAL)
  1 a 0
  2 c 1
  3 e 2
    (START)  |-  3
  0 b 1
  0 d 2

AUTHORS
    Darrell Raymond and Derick Wood

SEE ALSO
    xfm(5), fmreverse(1)

January 1994  Grail  2
NAME
xfmstar – compute '*' of an extended machine

SYNOPSIS

xfmstar xfm

xfmstar <xfm

DESCRIPTION

xfmstar computes '*' (Kleene closure) of xfm and writes the result on standard output.

xfmstar introduces no empty-string instructions. It first computes the '+' of xfm, then it clones the start state and makes it a final state.

xfm must conform to the Grail format for extended machines. Since every conventional machine is also an extended machine, xfmstar can be used to compute '*' of conventional machines.

EXAMPLES

% cat xfa3
(START) |- 0
0 a* 1
0 b* 2
0 c* 3
1 bb* 3
2 cc* 3
3 -| (FINAL)

% xfmstar xfa3
0 a* 1
0 b* 2
0 c* 3
1 bb* 3
2 cc* 3
3 -| (FINAL)
0 c* 0
1 bb* 0
2 cc* 0
4 a* 1
4 b* 2
4 c* 3
4 c* 0
% cat dfm2
1 a 2
1 a 3
1 a 4
2 - (FINAL)
3 - (FINAL)
4 - (FINAL)

% xfmstar <nfm2
1 a 2
1 a 3
1 a 4
2 - (FINAL)
3 - (FINAL)
4 - (FINAL)
1 a 1
5 a 2
5 a 3
5 a 4
5 a 1
5 - (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
xfm(5), xfmplus(1), fmstar(1)
NAME
xfmtore – convert an extended machine to a regular expression

SYNOPSIS
xfmtore xfm

xfmtore <xfm

DESCRIPTION
xfmtore computes a regular expression that accepts the same language as xfm, and writes the result on standard output.

xfmtore uses the state elimination method for producing the regular expression.

xfm must conform to the Grail format for extended machines. Since every conventional machine is also an extended machine, xfmtore can be used to convert conventional machines.

EXAMPLES

% cat xfm1
(START) |- 0
  0 ab* 1
  1 (c+d)* 2
  2 -| (FINAL)

% xfmtore <xfm1
ab*(c+d)*

% cat nfm2
(START) |- 1
  1 a 2
  1 a 3
  1 a 4
  2 -| (FINAL)
  3 -| (FINAL)
  4 -| (FINAL)

% xfmtore nfm2
a

% cat xfa3
(START) |- 0
  0 a* 1
xfmtore (1) User Commands xfmtore (1)

0 b* 2
0 c* 3
1 bb* 3
2 cc* 3
3 - (FINAL)

% xfmtore <xfm3
c*+a*bb*+b*cc*

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
xfm(5), re(5), fmtore(1), retofm(1)
NAME
xfmunion – compute the union of two extended machines

SYNOPSIS
xfmunion xfm1 xfm2

xfmunion xfm2 <xfm1

DESCRIPTION
xfmunion computes the union of xfm1 and xfm2. This is done by renumbering
the states of xfm2 and then simply appending its instructions to those of xfm1.

xfm1 and xfm2 must conform to the Grail format for extended machines. Since
every conventional machine is also an extended machine, xfmunion can also be
used to compute the union of conventional machines.

EXAMPLES

% cat xfa4
(START) |− 0
0 abc* 1
2 b+d 3
1 a(ef) 4
4 −| (FINAL)

% xfmunion xfa4 <xfm4
(START) |− 0
0 abc* 1
2 b+d 3
1 a(ef) 4
4 −| (FINAL)
(START) |− 5
5 abc* 6
7 b+d 8
6 a(ef) 9
9 −| (FINAL)

% cat nfm2
(START) |− 1
1 a 2
1 a 3
1 a 4
2 −| (FINAL)
xfmunion (1)  User Commands  xfmunion (1)

3 - | (FINAL)
4 - | (FINAL)

% xfmunion nfm2 xfa4
(START) | - 1
1 a 2
1 a 3
1 a 4
2 - | (FINAL)
3 - | (FINAL)
4 - | (FINAL)
(START) | - 5
5 abc* 6
7 b+d 8
6 a(e+f) 9
9 - | (FINAL)

AUTHORS
Darrell Raymond and Derick Wood

SEE ALSO
xfm(5), fmunion(1)