netscan **Reaction Network Finder**

Bertrand Clarke—University of British Columbia
Jay Mittenthal—University of Illinois at Urbana-Champaign
Glenn Fawcett—Caracal Consulting Inc.

Version 2.00

Version History

Version	Description
1.00	Initial program version
2.00	Restrictions on constraint input expressions removed.

1.0 Introduction

The `netscan` program accepts input files containing chemical reactions and overall goals (constraints) and outputs all subsets of the reactions, from smallest to largest, that satisfy all the constraints.

The following assumptions are made about reactions and the interconnections between them:

1. Reactions are an ongoing process; i.e., all reactions occur at the same time.

2. Molecules are either not present at all or present in essentially infinite amounts; i.e., the amount of each molecule used by a reaction is insignificant in comparison to the amount available from a constraint input or an earlier reaction output.

3. Reactions outputs are available to all other reactions; i.e, all reactions feed from and supply the same "molecular soup."

4. Reactions are specific; i.e., they ignore all molecules except their specified inputs.

2.0 Tutorial

Suppose we type in the command line:

```
netscan testfile >outfile
```

and we have a file called `testfile` that contains the following lines:

```
R1) A B -> D
R2) B C -> D
R3) C -> D E
B AND (A OR C) >> D
```

The file specifies three reactions, named "R1," "R2," and "R3."

- Reaction R1 accepts molecules "A" and "B" and produces molecule "D."

- Reaction R2 accepts molecules "B" and "C" and produces molecule "D."

- Reaction R3 accepts molecule "C" and produces molecules "D" and "E."

We could draw these reactions as:
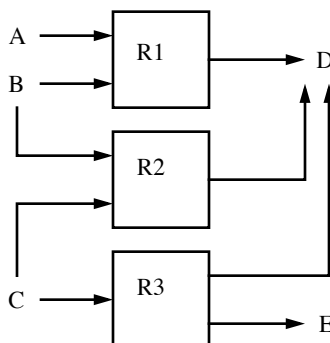
The file also specifies an overall constraint stating that molecule "D" must be produced if molecule "B" and either molecule "A" or "C" are provided to the reactions.

The program would then find, in this case, just one subset of the reactions that meets the constraint and would output the names of the reactions to the file `outfile` (the "`length =`" term refers to the number of reactions in the subset):

```
! Network #1 found, length = 2:
  R1 R2
```

You can see from the diagram above that both R1 and R2 are required to produce molecule "D" only for the "B AND (A OR C)" combination of molecules "A," "B," and "C." If R3 were included, the constraint would not be satisfied because providing "C" but not "A" or "B" to the reactions would produce "D."

The same output would be produced with two constraints:

```
B AND A >> D
B AND C >> D
```

This is because the subset consisting of R1 and R2 satisfies both constraints, taken individually. That is, considering only the molecules "B" and "A" (i.e., with "C" not present), the subset produces "D" only if both molecules "B" and "A" are provided to it. Considering only the molecules "B" and "C" (i.e., with "A" not

---

present), the subset produces "D" only if both molecules "B" and "C" are provided to it. No other subset satisfies both constraints.

Typical input files have far more reactions. In fact, there is no fixed limit to the number of reactions or constraints. The input file may also have one or more lines listing molecules that are always present; i.e., molecules that are available from a source other than a constraint. For example, to indicate to the program that molecules "G" and "H" are always present, a line can be included that says:

```
PRESENT G H
```

When many reactions produce the same output molecule, a very large number of output subsets may be produced. (If there are N reactions producing a molecule, there are 2^N-1 possible subsets of the reactions.) The program will require an amount of memory and time that doubles with each additional reaction, which starts to become significant as the number of reactions reaches 14 or so. Above that number the program will print a warning showing how many reactions have a common output molecule.

The following subsections show the output for various example inputs.

2.1 Example 1
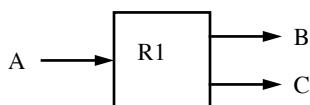

```
R1)  A -> B C
A >> B
```

Output is:

```
Status: deleting useless molecule C
! Network #1 found, length = 1:
  R1
```

Why: Molecule "C" is not mentioned in any constraint so the production of it is ignored.

2.2 Example 2
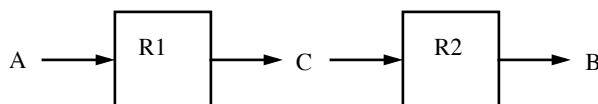

```
R1)  A -> C
R2)  C -> B
A >> B
```

Output is:

```
! Network #1 found, length = 2:
  R1 R2
```

Why: Molecule "A" leads to the production of molecule "B."

2.3 Example 3


```
R1)   A -> B
R2)   A -> C
A >> B
```

Output is:

```
Status: deleting useless reaction R2
Status: deleting useless molecule C
! Network #1 found, length = 1:
  R1
```

Why: Reaction R2 does not contribute to producing the molecules of the constraint and so is deleted.

2.4 Example 4


```
R1)   A -> B
R2)   C -> A
A >> B
```

Output is:

```
Status: deleting useless reaction R2
Status: deleting useless molecule C
! Network #1 found, length = 1:
  R1
```

Why: Molecule "C" is not mentioned in the constraint so reaction R2 is deleted.
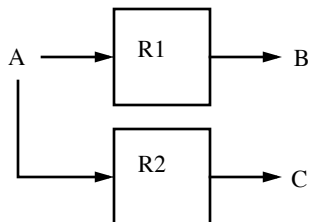
2.5 Example 5


```
R1)  A -> B
R2)  B -> A
A >> B
```

Output is:

```
! Network #1 found, length = 1:
  R1
! Network #2 found, length = 2:
  R1 R2
```

Why: Reaction R2 does not produce any "loose ends;" i.e., it does not create any unnecessary molecules. Thus it is not considered useless, even though it is not required.

2.6 Example 6
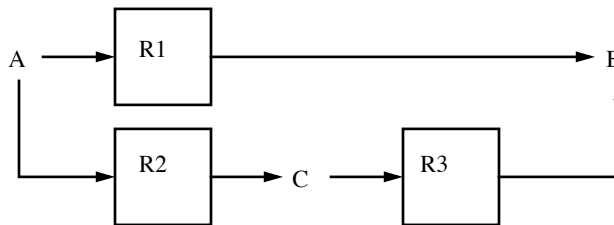

```
R1)  A -> B
R2)  A -> C
R3)  C -> B
A >> B
```

Output is:

```
! Network #1 found, length = 1:
  R1
! Network #2 found, length = 2:
  R2 R3
! Network #3 found, length = 3:
  R1 R2 R3
```

Why: The two parallel paths produce 2^2-1= 3 subsets.

2.7 Example 7

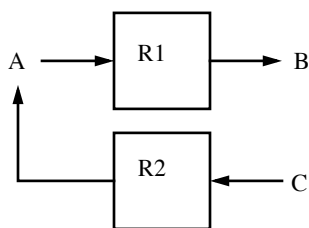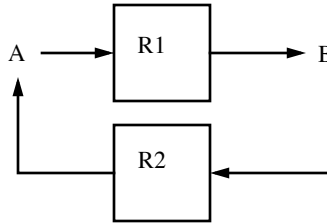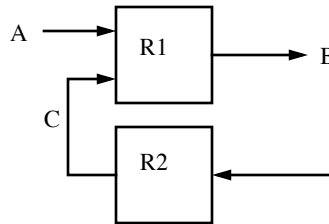```
R1) A C -> B
R2) B -> C
A >> B
```

Output is:

```
Status: deleting useless reaction R1
Status: deleting useless reaction R2
Status: deleting useless molecule C
Error: no effect from constraint input molecule A
Error: no generation of constraint output molecule B
Error: no possible satisfaction of constraint at testfile line 3
```

Why: The reactions cannot get started producing molecule "B" because molecule "C" (and thus "B") is required before "B" can be produced.

2.8 Example 8
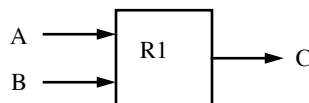

```
R1) A B -> C
A B >> C
```

Output is:

```
! Network #1 found, length = 1:
  R1
```

Why: Reaction R1 produces molecule "C" only if molecules "A" and "B" are provided to it.

2.9 Example 9
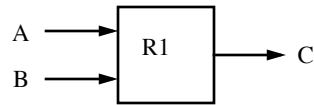

```
R1) A B -> C
A OR B >> C
```

Output is:

```
(no output)
```

Why: The reaction will not produce "C" if "A" or "B" is provided to it; both "A" and "B" must be provided before "C" is produced.

2.10 Example 10
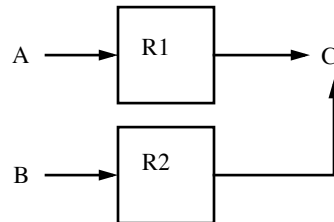

```
R1)  A -> C
R2)  B -> C
A OR B >> C
```

Output is:

```
! Network #1 found, length = 2:
  R1 R2
```

Why: The reactions R1 and R2 will produce "C" if "A" or "B" is provided to them.

2.11 Example 11
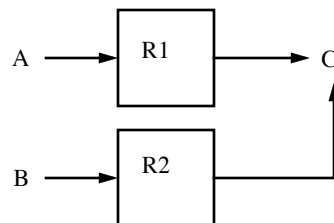
```
R1)  A -> C
R2)  B -> C
A B >> C
```

Output is:

```
(no output)
```

Why: There is no combination of R1 and R2 that will produce molecule "C" only if both molecule "A" and molecule "B" are provided.

2.12 Example 12
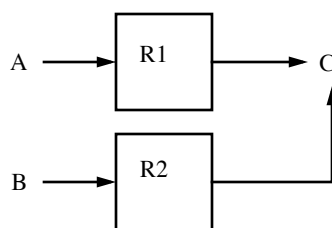

```
R1)  A -> C
R2)  B -> C
A >> C
```

Output is:

```
Status: deleting useless reaction R2
Status: deleting useless molecule B
! Network #1 found, length = 1:
  R1
```

Why: Molecule "B" is never provided so reaction R2 is useless.

2.13 Example 13
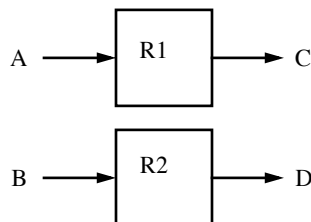

```
R1)  A -> C
R2)  B -> D
A B >> C D
```

```
! Network #1 found, length = 2:
  R1 R2
```

Why: The reactions will produce both "C" and "D" when both "A" and "B" are provided to it. While it is true that "D" is produced if "B" is provided, the constraint only specified what happens for both "C" and "D" taken together.

2.14 Example 14
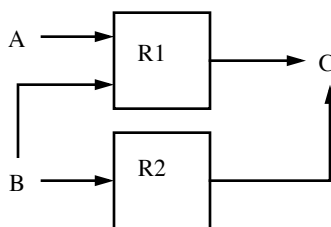

```
R1) A B -> C
R2) B -> C
A B >> C
```

Output is:

```
! Network #1 found, length = 1:
  R1
```

Why: The program considered all three combinations of R1 and R2 and only R1, by itself, produced "C" only when both "A" and "B" were provided.

2.15 Example 15
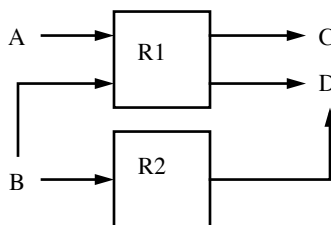

```
R1) A B -> C D
R2) B -> D
A B >> C D
```

Output is:

```
! Network #1 found, length = 1:
  R1
! Network #2 found, length = 2:
  R1 R2
```

Why: Both "C" and "D" are only produced if "A" and "B" are provided. While it is true that "D" is produced if "B" is provided, the constraint only specified what happens for both taken together. On the other hand, the constraints:

```
A B >> C
A B >> D
```

would produce the single subset R1 because the constraints specify that "C" and "D" are each only produced when both "A" and "B" are provided.

2.16 Example 16
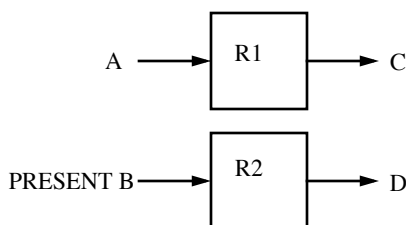

```
R1) A -> C
R2) B -> D
PRESENT B
A >> C D
```

Output is:

```
! Network #1 found, length = 2:
  R1 R2
```

Why: Again, although "D" is always produced, this is irrelevant. The constraint specifies that *both* "C" and "D" together will be produced only when "A" is provided.

2.17 Example 17
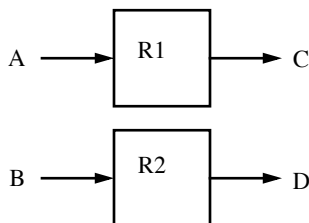

```
R1) A -> C
```

```
R2) B -> D
A >> C D
B >> C D
```

Output is:

```
(no output)
```

Why: Looking at the first constraint by itself, there is no subset of the reactions that will produce both "C" and "D" only when "A" is provided.

2.18 Example 18
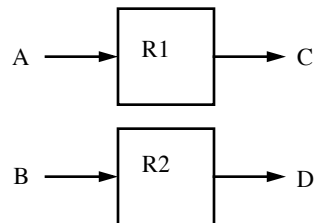

```
R1) A -> C
R2) B -> D
A >> C
B >> D
```

Output is:

```
! Network #1 found, length = 2:
  R1 R2
```

Why: Taken individually, both constraints are satisfied although the resulting subset of reactions is disjoint.

2.19 Example 19
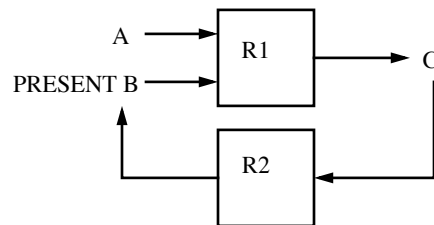

```
R1) A B -> C
R2) C -> B
PRESENT B
A >> C
```

Output is:

```
! Network #1 found, length = 1:
  R1
! Network #2 found, length = 2:
  R1 R2
```

Why: Only R1 is required but adding R2 replenishes the supply of PRESENT molecule "B" and is thus not considered useless.

If the `-r` flag is provided when the program is invoked (see Section 3.0), reactions are considered useless if they only feed PRESENT molecules:

```
Status: deleting useless reaction R2
! Network #1 found, length = 1:
  R1
```

2.20 Example 20
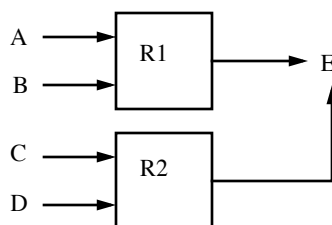

```
R1) A B -> E
R2) C D -> E
A B >> E
C D >> E
```

Output is:

```
! Network #1 found, length = 1:
R1 R2
```

Why: Taken individually, both constraints are satisfied. Note that when the first constraint is tested, molecules "C" and "D" are assumed to be not present. Similarly, when the second constraint is tested, molecules "A" and "B" are assumed to be not present.

3.0 Command Line

The `netscan` program is invoked from the command line, either from the shell in Unix, Linux or Mac OS X or from a DOS command line under Windows. Format of the command line is:

```
netscan [-flags]... infile...  >outfile
```

For example:

```
netscan -m myfile1.rxl myfile2.rxl >myfile.out
```

Typing "`netscan -help`" will provide the following useful message:

```
Usage: netscan [-flags]... infile...  >outfile

 -help  print helpful message

 --  read standard input within input file list
 -e  echo significant input to output
 -m  place messages in output, as comments
 -r  delete reactions that replenish PRESENT molecules
 -s  don't show status messages

 -c  to debug, show searching for candidates
 -d  to debug, don't combine duplicate reactions
 -l  to debug, show lexical analyzer output
 -p  to debug, show parser output
 -u  to debug, don't delete useless reactions
 -x  to debug, show constraint test cases/tests
 -z  to debug, show memory before/after searching
```

The flags commonly used are:

Flag	Function
--	Read input from the standard input. For example, the list of files "inputfile1 -- inputfile2" means read from inputfile1 then from the standard input then from inputfile2.
-e	The content of the input file(s) (except blank lines and comments) is placed in the output, preceded by an exclamation mark at the start of the line.
-m	Messages, such as status messages, warnings and errors, are placed in the output file, preceded by and exclamation mark, as well as being written to the normal error output.
-r	Reactions that only replenish PRESENT molecules are considered useless and are deleted before searching for reaction subsets.
-s	Status messages are not shown, only warnings and errors.

4.0 Input File

Input files are specified when the program is invoked (see Section 3.0).

4.1 File Lines

The input file is a text file and consists of lines, each terminated by either a Unix, Windows or Macintosh newline. Blank lines are allowed and ignored. All lines must be less than 512 characters in length, not counting comments. The order of input lines has no effect on the program output.

4.2 Comments

Any occurrence of the exclamation point "!" indicates the start of a comment, which continues to the end of the line. Comments are for human convenience and are deleted by the program before further processing.

4.3 Case Insensitivity

All words are case insensitive; that is, two words will be considered to match if they contain the same sequence of letters and symbols regardless of the case of the letters. All reaction and molecule names are words, as are the strings "AND," "OR" and "PRESENT."

4.4 Whitespace

Any string of one or more spaces or tabs are considered whitespace. Any whitespace at the beginning or end of a line is deleted before further processing. Some whitespace must be placed between words in order to indicate word boundaries but any amount may be used. Whitespace may be placed around operators such as "+" or "->" or ">>" or "(" or ")" if desired.

4.5 Molecule and Reaction Names

The names "and" and "or" and "present" may not be used, nor may a name contain whitespace or commas or the strings ")", "(", "+", "->", "l", ">>" or "!". Names may contain any sequence of letters, numbers or special characters such as "&", ":", """, ".", "_", "-", or "*". Examples of valid names are:

- `A&:A*`

- `pB.A&`

- `areallylongnameconsistingof64charactersfollowedbyaquotecharacter'`

4.6 AND Separators

An "and-list" is a list of one or more molecules considered as a group, for example:

```
MOLA AND MOLB AND MOLC
```

Instead of the word "AND", plus signs or commas or just spaces may be used. The following are all equivalent to the above:

```
MOLA MOLB MOLC
MOLA, MOLB, MOLC
MOLA+MOLB+MOLC
MOLA and MOLB+MOLC
MOLA+MOLB MOLC
```

4.7 Reactions

A reaction is defined on a single line of the input file. The format of a reaction line is:

```
react-name ) input-and-list -> output-and-list | input-list-continued
```

For example:

```
R1) MOLA MOLB -> MOLC MOLD
```

defines a reaction named "R1" that accepts MOLA and MOLB and produced MOLC and MOLD. The following is interpreted to be exactly the same:

```
R1) MOLA -> MOLC MOLD | MOLB
```

The reaction name is optional. If omitted (along with the following right parenthesis), the program will assign the reaction a name of the form "inputfilename_line_33." This may be useful for simple tests.

The same molecule may not be in both the input and output of a reaction.

There is no fixed limit on the number of reactions. However a maximum of 32 non-identical, non-useless reactions may produce the same output molecule. This is because the number of candidate reactions increases exponentially with the number of reactions producing a molecule.

4.8 Constraints

A constraint is defined on a single line of the input file. The format of a constraint line is:

```
input-expression >> output-and-list
```

There is no fixed limit on the number of constraints.

The same molecule may not be in both the input and output of a constraint.

4.9 Constraint Expressions

The input expression of a constraint may consist of molecule names, parentheses, and-lists and or-lists, for example:

```
(MOLA AND MOLB) OR (MOLC AND MOLD)
MOLA MOLB
(MOLA)(MOLB)
(MOLA OR MOLB)(MOLC or MOLD)
```

When both and-lists and or-lists are used without parentheses, "AND" binds tighter than "OR," for example

```
MOLA AND MOLB OR MOLC or MOLD
```

is equivalent to the clearer

```
(MOLA AND MOLB) OR (MOLC AND MOLD)
```

4.10 Present Lists

A present list is defined on a single line of the input file, although multiple present lists may be in the file. The format of a present list line is:

```
PRESENT present-and-list
```

For example:

```
PRESENT MOLA MOLB
```

The program will generate a warning if a molecule is mentioned in more than one present list.

5.0 Output File

Provided that the debugging flags are not used, the output file consists of comments and reaction subsets.

Comments start with an exclamation mark. They are intended to be readable by humans.

A subset is a set of reaction names, separated by spaces. A large subset may continue over several lines. After the last line of a subset, the program generates a blank line.

For example, the program might output:

```
! Network #1 found, length = 1:
  R1
```

(blank line)

```
! Network #2 found, length = 2:
  R1 R3
```

(blank line)

(end of file)

5.1 Duplicate Reactions

Reactions that have the same inputs and outputs are considered duplicates. The program combines them immediately so they will not create extra subsets to search (recall that if there are N reactions producing a molecule, there are 2^N-1 possible subsets).

When the program outputs a reaction subset, it outputs the duplicate names together, surrounded by parentheses. For example, the subset

```
R1 (R2 R3) R4
```

has duplicate reactions R2 and R3. The single line above is shorthand for the 2^2-1 = 3 subsets:

```
R1 R2 R4
```

```
R1 R3 R4
```

```
R1 R2 R3 R4
```

A subset such as

```
R1 (R2 R3 R4 R5) (R6 R7 R8) R9
```

would be shorthand for 15 x 7 = 105 different subsets.

6.0 Theory of Operation

The program uses seven processing steps to generate its output:

1. Lexical analyzing of the input

2. Parsing the resulting tokens

3. Combining duplicate reactions

4. Connectivity testing and eliminating useless reactions

5. Estimating complexity

6. Generating candidate subsets

7. Testing constraint logic

Each step is described in detail below.

6.1 Lexical Analyzing

Lexical analyzing breaks the input files into tokens (strings of characters that are interpreted as a name or operator) and ignores extra whitespace and comments. For example, the input line:

```
(a+b )>>f! a comment
```

would produce the token output:

```
TOK_LPAREN = "("
TOK_ID = "a"
TOK_AND = "+"
TOK_ID = "b"
TOK_RPAREN = ")"
TOK_DBLARROW = ">>"
TOK_ID = "f"
```

6.2 Parsing

Parsing determines the semantic meaning of the input tokens produced by the lexical analyzer. For example, the input file lines:

```
R1)a+b+c->f
(((a)or(b))c)>>f
```

Would produce the parser output:

```
Reaction "R1":
  Input  molecule: a
  Input  molecule: b
  Input  molecule: c
  Output molecule: f
Constraint "testfile line 2":
  Input expression:
    (AND (OR  a b ) c )
  Output molecule: f
```

The parser also simplifies constraint expressions into and-lists and or-lists, as can be seen above.

6.3 Combining Duplicate Reactions

Duplicate reactions are combined immediately after the reactions are stored in memory. An input file:

```
R1) a -> b
R2) b -> c
R3) b -> c
R4) b -> c
R5) c -> d
a >> d
```

Produces the output:

```
! Network #1 found, length = 3:
  R1 ( R2 R3 R4 ) R5
```

6.4 Connectivity Testing

Connectivity testing is used to eliminate useless reactions quickly and early, before they can slow down subset generation. For example, the input file:

```
R1) a -> b
R2) b -> c
R3) c -> d
b >> c
```

produces the output:

```
Status: deleting useless reaction R1
Status: deleting useless reaction R3
Status: deleting useless molecule a
Status: deleting useless molecule d
! Network #1 found, length = 1:
  R2
```

The connectivity testing uses the following rules:

- A molecule is considered "input-connected to a constraint" if it is PRESENT or if it is found in the input expression of the constraint or if any of the reactions that generate it are input-connected to the same constraint.

- A reaction is considered "input-connected to a constraint" if all of its input molecules are input-connected to the same constraint.

- A molecule is considered "output-connected to a constraint" if it is found in the output list of the constraint or if any reaction that uses it is output-connected to the same constraint. (If the -r flag is given on the command line, a molecule that is PRESENT is not considered output connected to any constraint.)

- A reaction is considered "output-connected to a constraint" if any of its output molecules are output-connected to the same constraint.

The program recursively solves the above rules for all reactions and molecules until the results stabilize. (Connectivity for all constraints is determined in parallel.) Then:

- A reaction is considered useless there is no constraint for which it is both input-connected and output-connected.

All useless reactions are deleted. Then the connectivity testing is done again until no more useless reactions are produced. Then:

• A molecule is considered useless if it is not in the input expression or output list of any constraint and it does not feed any reaction.

All useless molecules are deleted.

6.5 Estimating Complexity

After deleting useless reactions and molecules, the program estimates the complexity of the input in order to estimate the time and memory required to search all subsets. These estimates are then output as status messages, for example:

```
Status: estimated memory required = 10.2 Mb
Status: estimated time to build test cases = 4.3 seconds
Status: estimated time to search = 12 minutes
```

When searching reaction subsets, the program works backwards from the constraint outputs. The complexity estimate uses a slightly less accurate, but faster, method:

• The complexity is initially set to the number of reactions.

• Molecules are processed in an arbitrary order. Molecules that do not feed any reaction and are not in the output list of any constraint are ignored. When a molecule is processed, the complexity is multiplied by $2^P - 1$, where P is the number of reactions, not yet counted, which produce the molecule. (Once a reaction is counted in this way, it is not counted again for another molecule.)

The memory requirement is estimated by multiplying the complexity by the size of an internal structure called a node. The time is estimated by multiplying the complexity by a constant determined by experiment.

Test case generation is described in Section 6.7. The time to generate test cases is calculated by adding up the value 2^E for each constraint, where E is the number of different molecules in the constraint expression. This sum is multiplied by the number of molecules used anywhere and a constant determined by experiment.

6.6 Generating Candidate Subsets

The program uses a breadth-first search to find all subsets of reactions from smallest to largest.

To perform the search, the program uses variable-sized structures called "searchsets," which are built up of fixed-sized structures called nodes. (Using fixed-sized structures eliminates memory fragmentation, which is a problem for breadth-first searching.)

Each searchset contains:

• a list of reactions that comprise the current subset

• a list of reactions that are disallowed

• a list of molecules to be searched

Initially, there is one searchset, which has no reactions on either reaction list. Its list of molecules contains the molecules that are in the output lists of all constraints, with no duplicates.

The generation of candidate subsets uses the following algorithm:

1. The searchset with the smallest number of reactions in its current subset is processed (in the case of a tie, an arbitrary one is picked). If there are no searchsets, the program terminates.

2. The searchset's current reaction subset list is examined. If, for every constraint, the reactions use all the molecules in the constraint input expression and outputs all of the molecules in the constraint output list, the subset is said to satisfy all the constraints. If so, and if all the molecules on searchset's search list are

either PRESENT or produced by a reaction in the current subset or are in a constraint input expression, the subset is sent for further processing (see Section 6.7).

3. If there are no molecules on the searchset's search list, the searchset is deleted. Otherwise, the molecule on the search list with the least number of reactions generating it in the input file is removed and examined to find the number of reactions generating it that are not on one of the searchset's two reaction lists. If there are N such new reactions, 2^N-1 searchsets will replace the searchset being examined (as discussed below). Otherwise, if there are no new reactions, the next molecule is examined.

4. A searchset is created for all subsets of the new reactions. Each new searchset will be identical to the old searchset but will have added all the new reactions, either on its current subset list or on its disallowed list. (For example, if the new reactions were R4 and R5, three new searchsets will replace the old one. The first will have R4 added to its current subset and R5 added to its disallowed list. The second will have R5 added to its current subset and R4 added to its disallowed list. The third will have R4 and R5 added to its current subset and nothing added to its disallowed list.) The inputs molecules of the reactions added to the current subset are added to the search list. The search then continues at step 1.

For example, consider the input file:

```
R1) a b -> c d
R2) a -> c
R3) b -> d
a b >> c d
```

Initially, one searchset is created with molecules "c" and "d" to search:

```
Initial searchset 0x0414c5b8 created
Searchset 0x0414c5b8 active:
  Reactions (0):
  Disallowed reactions:
  Molecules yet to search: c d
  Satisfaction bitmap: 0000000f
```

Both "c" and "d" have two reactions generating them so molecule "c" is arbitrarily chosen:

```
Searchset 0x0414c5b8 checking c
```

Molecule "c" is produced by two new reactions, requiring three searchsets to replace the old. Two new searchsets are created, in addition to the original.

```
Searchset 0x0414c578 created
Searchset 0x0414c518 created
```

One of the searchsets with only one reaction in the current subset is arbitrarily chosen:

```
Searchset 0x0414c578 active:
  Reactions (1): R1
  Disallowed reactions: R2
  Molecules yet to search: b a d
  Satisfaction bitmap: 00000000
```

Using bitmap arithmetic, the searchset is found to satisfy all of the constraints (there is only one). The candidate subset is passed on for logic testing (which decides to output it):

```
Searchset 0x0414c578 testing constraint logic
! Network #1 found, length = 1:
  R1
```

Molecules "b" and "a" are examined without finding any new reactions:

```
Searchset 0x0414c578 checking b
Searchset 0x0414c578 checking a
```

Molecule "d" is examined and reaction R3 is added to the current subset:

```
Searchset 0x0414c578 checking d
```

The next searchset with only one reaction is chosen but it does not satisfy the constraint ("a" and "d" are not connected):

```
Searchset 0x0414c518 active:
  Reactions (1): R2
  Disallowed reactions: R1
  Molecules yet to search: a d
  Satisfaction bitmap: 0000000a
```

Molecule "a" is examined without finding any new reactions:

```
Searchset 0x0414c518 checking a
```

Molecule "d" is examined and reaction R3 is added to the current subset:

```
Searchset 0x0414c518 checking d
```

The next searchset (with two reactions) is chosen:

```
Searchset 0x0414c5b8 active:
  Reactions (2): R2 R1
  Disallowed reactions:
  Molecules yet to search: b a d
  Satisfaction bitmap: 00000000
```

The searchset is found to satisfy the constraint. The candidate subset is passed on for logic testing (which decides to output it):

```
Searchset 0x0414c5b8 testing constraint logic
! Network #2 found, length = 2:
  R1 R2
```

Molecules "b" and "a" are examined without finding any new reactions:

```
Searchset 0x0414c5b8 checking b
Searchset 0x0414c5b8 checking a
```

Molecule "d" is examined and reaction R3 is added to the current subset:

```
Searchset 0x0414c5b8 checking d
```


The next searchset (with two reactions) is chosen:

```
Searchset 0x0414c578 active:
  Reactions (2): R3 R1
  Disallowed reactions: R2
  Molecules yet to search: b
  Satisfaction bitmap: 00000000
```

The searchset is found to satisfy the constraint. The candidate subset is passed on for logic testing (which decides to output it):

```
Searchset 0x0414c578 testing constraint logic
! Network #3 found, length = 2:
```

```
  R1 R3
```

Molecule "b" is examined without finding any new reactions and there are no molecules left so the searchset is deleted:

```
Searchset 0x0414c578 checking b
Searchset 0x0414c578 deleted
```

The next searchset (with two reactions) is chosen:

```
Searchset 0x0414c518 active:
  Reactions (2): R3 R2
  Disallowed reactions: R1
  Molecules yet to search: b
  Satisfaction bitmap: 00000000
```

The searchset is found to satisfy the constraint. The candidate subset is passed on for logic testing (which decides to output it):

```
Searchset 0x0414c518 testing constraint logic
! Network #4 found, length = 2:
  R2 R3
```

Molecule "b" is examined without finding any new reactions and there are no molecules left so the searchset is deleted:

```
Searchset 0x0414c518 checking b
Searchset 0x0414c518 deleted
```

The next searchset (with three reactions) is chosen:

```
Searchset 0x0414c5b8 active:
  Reactions (3): R3 R2 R1
  Disallowed reactions:
  Molecules yet to search: b
  Satisfaction bitmap: 00000000
```

The searchset is found to satisfy the constraint. The candidate subset is passed on for logic testing (which decides to output it):

```
Searchset 0x0414c5b8 testing constraint logic
! Network #5 found, length = 3:
  R1 R2 R3
```

Molecule "b" is examined without finding any new reactions and there are no molecules left so the searchset is deleted:

```
Searchset 0x0414c5b8 checking b
Searchset 0x0414c5b8 deleted
```

There are no more searchsets so the program terminates.

6.7 Test Case Generation for Constraint Logic

The candidate subset generation in the previous section only checks that a subset uses all the constraint inputs and generates all the constraint outputs. Constraint logic testing guarantees that only subsets that produce the logic required by the constraints will be output.

There are two phases to this testing:

- Before any candidates are generated, the program generates a set of test cases for each constraint.

- As each candidate is generated, its logic is tested against all the test cases.

Each test case is a set of inputs to the subset along with an expected output. For example, consider the constraint:

```
(a and b) or (c and d) or e >> f
```

The parsed output of the expression is:

```
(OR  (AND a b ) (AND c d ) e )
```

The program generates the following test cases to prove the logic:

```
e=1 d=0 c=0 b=0 a=0 OUTPUT=1
e=0 d=1 c=1 b=0 a=0 OUTPUT=1
e=0 d=1 c=0 b=1 a=0 OUTPUT=0
e=0 d=0 c=1 b=1 a=0 OUTPUT=0
e=0 d=1 c=0 b=0 a=1 OUTPUT=0
e=0 d=0 c=1 b=0 a=1 OUTPUT=0
e=0 d=0 c=0 b=1 a=1 OUTPUT=1
```

In the table above, the value "1" indicates TRUE and "0" indicates false. "OUTPUT=1" means that all the molecules in the output list of the constraint must be produced and "OUTPUT=0" means that not all of the molecules in the output list may be produced.

These cases were generated by first generating all 2^E possible input cases for the constraint, where E is the number of different molecules in the input expression of the constraint. In this example, 32 are generated. However, because a netowr can only implement logical OR and AND operations, not logical NOT operation, the number of test cases can often be reduced. The program eliminates test cases with OUTPUT=1 if their inputs are a superset of some other case with OUTPUT=1. Similarly, the program eliminates test cases with OUTPUT=0 if their inputs are a subset of some other case with OUTPUT=0. In the above example, only 7 of the 32 potential test cases remain.

6.8 Testing Constraint Logic

During candidate testing, the molecule inputs to the reactions in the subset are set to the values in the test case being considered. PRESENT molecules are set to one and molecules not in the test case are set to zero. The reactions are simulated until the molecule values stabilize and then the constraint's output molecules are examined to determine if together they match the test case's output value.

Each test case must be run through the reaction subset (and there will be two or more for each constraint). Actually, the program processes all test cases of all constraints simultaneously. Each molecule has a bitmap, which is as wide as the number of test cases, to allow all cases to be solved independently yet together.

7.0 Compiling the Source Code

The source code consists of nearly 2,800 ANSI C statements separated into 15 files:

File	Function
`ns_defs.h`	Global structure and constant definitions
`ns_proto.h`	Function prototype definitions
`ns_main.c`	Main entry point, command line parsing, help
`ns_lexical.c`	Lexical analysis (see Section 6.1)
`ns_parse.c`	Parsing (see Section 6.2)
`ns_netstore.c`	Error checking and storing of input file, converting to structures, and duplicate reaction combining (see Section 6.3)
`ns_connect.c`	Connectivity testing and useless reaction/molecule elimination (see Section 6.4)
`ns_complexity.c`	Complexity estimation (see Section 6.5)
`ns_candidates.c`	Candidate subset generation (see Section 6.6)
`ns_expr.c`	Constraint expression test case generation and testing (see Section 6.7)
`ns_output.c`	Output file formatting (see Section 5.0)
`ns_file.c`	File handling
`ns_memory.c`	Memory allocation/deallocation
`ns_listops.c`	Variable-length list operations
`ns_bitmaps.c`	Variable-length bitmap operations

The code is POSIX compliant and should work on 64-bit machines. It can be compiled with any ANSI C compiler, possibly after converting the UNIX line endings to the native form of the computer used for compiling. For example, the GNU C compiler can be used to produce an executable program using the following command line:

```
gcc *.c
```

The $-lm$ flag may be required on some systems to load the math library.

Binary versions for the Windows DOS command line and Macintosh Classic environment have been provided.

8.0 Test Files

Six test input files have been provided to allow user-compiled programs to be checked.

8.1 pons_asinorum.rxl

The most basic program that produces a load on the processor, this file takes 10 seconds or more to generate its output. It consists of 16 parallel reaction paths:

```
R1)   b -> c

R2)   c -> q1
R3)   q1 -> d

R4)   c -> q2
R5)   q2 -> d

R6)   c -> q3
R7)   q3 -> d


(etc., etc.)


R32) c -> q16
R33) q16 -> d

R34) d -> e

b >> e
```

There are 65,535 output subsets generated:

```
! Network #1 found, length = 4:
  R1 R2 R3 R34

! Network #2 found, length = 4:
  R1 R4 R5 R34

! Network #3 found, length = 4:
  R1 R6 R7 R34

(etc., etc.)

! Network #65534 found, length = 32:
  R1 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R20 R21
  R22 R23 R24 R25 R26 R27 R28 R29 R30 R31 R32 R33 R34

! Network #65535 found, length = 34:
  R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19
  R20 R21 R22 R23 R24 R25 R26 R27 R28 R29 R30 R31 R32 R33 R34
```

`Paths18.rxl` is the same, but with 18 different paths. It requires approximately four times as long to search, uses about 80 Mb of memory, and generates an output file that is 123 Mb in size.

8.2 4_test.rxl

This file is a basic test of alternate reaction paths and duplicate elimination:

```
R1) a b -> c d
R2) a -> c
R3) b -> d
R4) a -> c
a b >> c d
```

It produces four subsets:

```
Status: combining duplicate reaction R4
! Network #1 found, length = 1:
  R1
! Network #2 found, length = 2:
  R1 ( R2 R4 )
! Network #3 found, length = 2:
  R1 R3
! Network #4 found, length = 2:
  ( R2 R4 ) R3
! Network #5 found, length = 3:
  R1 ( R2 R4 ) R3
```

8.3 useless.rxl

This file tests duplicate elimination. All but one reaction is eliminated because there is no source for the molecule xx:

```
R1)  a        -> b

R2)  b        -> q1
R3)  q1  xx -> r1
R4)  r1       -> c

R5)  b        -> q2
R6)  q2  xx -> r2
R7)  r2       -> c

R8)  b        -> q3
R9)  q3  xx -> r3
R10) r3       -> c
```

(etc., etc.)

```
R71) b        -> q24
R72) q24 xx -> r24
R73) r24      -> c

R74) c -> d
R75) a -> d

     a >> d
```

The output should be:

```
Status: deleting useless reaction R1
Status: deleting useless reaction R2
Status: deleting useless reaction R3
(etc., etc.)
Status: deleting useless reaction R73
Status: deleting useless reaction R74
Status: deleting useless molecule b
Status: deleting useless molecule q1
Status: deleting useless molecule xx
Status: deleting useless molecule r1
Status: deleting useless molecule c
Status: deleting useless molecule q2
Status: deleting useless molecule r2
(etc., etc.)
Status: deleting useless molecule q24
Status: deleting useless molecule r24
! Network #1 found, length = 1:
  R75
```

8.4 duplicates.rxl

This file demonstrates duplicate elimination; no program could solve this network without it. It consists of 192 reactions: eight reactions duplicated 24 times.

```
R1)   a b -> q r
R2)   c d -> s t
R3)   e f -> u v
R4)   g h -> w x
R5)   q s -> i j
R6)   u w -> k l
R7)   r t -> m n
R8)   v x -> o p

R9)   a b -> q r
R10)  c d -> s t
R11)  e f -> u v
R12)  g h -> w x
R13)  q s -> i j
R14)  u w -> k l
R15)  r t -> m n
R16)  v x -> o p

(etc., etc.)

R185) a b -> q r
R186) c d -> s t
R187) e f -> u v
R188) g h -> w x
R189) q s -> i j
R190) u w -> k l
R191) r t -> m n
R192) v x -> o p
```

```
      a b c d e f g h >> i j k l m n o p
```

The output produces a single subset with many duplicate reactions:

```
Status: combining duplicate reaction R9
Status: combining duplicate reaction R17
Status: combining duplicate reaction R25
Status: combining duplicate reaction R33
(etc., etc.)
Status: combining duplicate reaction R176
Status: combining duplicate reaction R184
Status: combining duplicate reaction R192
! Network #1 found, length = 8:
  ( R1 R9 R17 R25 R33 R41 R49 R57 R65 R73 R81 R89 R97 R105 R113 R121
  R129 R137 R145 R153 R161 R169 R177 R185 ) ( R2 R10 R18 R26 R34 R42
  R50 R58 R66 R74 R82 R90 R98 R106 R114 R122 R130 R138 R146 R154 R162
  R170 R178 R186 ) ( R3 R11 R19 R27 R35 R43 R51 R59 R67 R75 R83 R91 R99
  R107 R115 R123 R131 R139 R147 R155 R163 R171 R179 R187 ) ( R4 R12 R20
  R28 R36 R44 R52 R60 R68 R76 R84 R92 R100 R108 R116 R124 R132 R140 R148
  R156 R164 R172 R180 R188 ) ( R5 R13 R21 R29 R37 R45 R53 R61 R69 R77
  R85 R93 R101 R109 R117 R125 R133 R141 R149 R157 R165 R173 R181 R189 )
  ( R6 R14 R22 R30 R38 R46 R54 R62 R70 R78 R86 R94 R102 R110 R118 R126
  R134 R142 R150 R158 R166 R174 R182 R190 ) ( R7 R15 R23 R31 R39 R47
  R55 R63 R71 R79 R87 R95 R103 R111 R119 R127 R135 R143 R151 R159 R167
  R175 R183 R191 ) ( R8 R16 R24 R32 R40 R48 R56 R64 R72 R80 R88 R96 R104
  R112 R120 R128 R136 R144 R152 R160 R168 R176 R184 R192 )
```

8.5 expr.rxl

This file tests a variety of different constraint input expressions, requiring 94 test cases (which may be seen using the **-x** command line option):

```
R1) a b -> e
R2) c d -> e
R3) e f -> i
R4) g h -> i
R5) i j -> k
R6) k l -> m
R7) m n -> o
R8) p q -> r
R9) s -> u
R10) t -> u


(a b) or (c d) >> e
(((a b) or (c d)) f) or (g h) >> i
((((a b) or (c d)) f) or (g h)) j >> k
((((a b) or (c d)) f) or (g h)) j l >> m
((((a b) or (c d)) f) or (g h)) j l n >> o
p q >> r
((a b) or (c d)) p q >> e r
s or t >> u
p (s or t) q >> u r
((a b) or (c d)) (s or t) p q >> e r u
```

A single valid subset is found, consisting of all 10 reactions:

```
! Network #1 found, length = 10:
  R1 R2 R3 R4 R5 R6 R7 R8 R9 R10
```

8.6 varied.rxl

This file is a complex set of reactions that tests many special cases of molecular production and use:

```
R1) A B -> C II
R2) II -> JJ
R3) N P1 -> E EE | C
PRESENT P1
R4) QQ -> N NN
R5) P2 -> I PP | E
PRESENT P2
R6) EE X -> G GG
R7) PP -> Z
R8) P3 -> OO | PP
PRESENT P3
R9) I -> K
R10) G GG N -> P
R11) P -> HH LL
R12) Z -> BB | AA
R13) AA OO -> CC
R14) P4 -> FF | OO
PRESENT P4
R15) P5 -> R MM | P
PRESENT P5
R16) BB -> DD
R17) CC -> DD
R18) P6 -> DD | KK
PRESENT P6
R19) P7 -> X | MM
PRESENT P7
R20) DD -> U
R21) K N -> T U
R22) R U -> W
R23) KK -> R
R24) L -> QQ
R25) I -> K

PRESENT MM      ! remove recurrence
A B L >> W
```

The output is a single subset with 12 reactions in a chain:

```
Status: combining duplicate reaction R25
Status: deleting useless reaction R2
Status: deleting useless reaction R7
Status: deleting useless reaction R8
Status: deleting useless reaction R11
Status: deleting useless reaction R12
Status: deleting useless reaction R13
Status: deleting useless reaction R14
Status: deleting useless reaction R16
Status: deleting useless reaction R17
Status: deleting useless reaction R18
```

```
Status: deleting useless reaction R20
Status: deleting useless reaction R23
Status: deleting useless molecule II
Status: deleting useless molecule JJ
Status: deleting useless molecule NN
Status: deleting useless molecule PP
Status: deleting useless molecule Z
Status: deleting useless molecule P3
Status: deleting useless molecule OO
Status: deleting useless molecule HH
Status: deleting useless molecule LL
Status: deleting useless molecule BB
Status: deleting useless molecule AA
Status: deleting useless molecule CC
Status: deleting useless molecule P4
Status: deleting useless molecule FF
Status: deleting useless molecule DD
Status: deleting useless molecule P6
Status: deleting useless molecule KK
Status: deleting useless molecule T
! Network #1 found, length = 12:
  R1 R3 R4 R5 R6 ( R9 R25 ) R10 R15 R19 R21 R22 R24
```