Daylight Chemical Information Systems, Inc.
120 Vantis - Suite 550 - Aliso Viejo, CA 92656
tel +1 949-831-9990 - fax +1 949-831-9902 - www.daylight.com

Daylight Theory Manual

Table of Contents

Table of Contents

Daylight Theory Manual

Daylight Version 4.9
Release Date 02/01/08
Daylight Chemical Information Systems, Inc.

1. Introduction

This document presents the foundations of the Daylight Chemical Information System, including our motivations and goals, theoretical discussions on chemical information processing and chemical database design, and insight into the inner workings of some of our algorithms.

At Daylight, our goal is to provide the best known computer algorithms for chemical information processing to those who need them. We provide these algorithms both as a Daylight Toolkit programmer's library and as a set of ready-to-use programs for the non-programmer. This document is meant to serve both groups as well as users of other products that incorporate Daylight's technology.

This document does not provide specifics on how to use Daylight applications and toolkits (see User Manual's, Programmer's Guide, and the man pages for this information). Rather, it is designed to help you understand how the software works. In general each chapter is written with the assumption that the reader has read the previous chapters.

2. Molecules and Reactions in A Computer

There are atoms and space. Everything else is opinion. -- *Democritus*

The foundation of a chemical information system is its ability to represent molecules in a computer and to communicate a molecule's structure from one place to another. This can seem like a simple problem at first glance so that easy solutions are often proposed and implemented. But a close examination of the problem reveals that several subtle traps await the unwary and methods of avoiding them must be considered before an effective computer representation of a molecule can be designed.

2.1 Representing Molecules

To represent a molecule in a computer, we must first choose a particular physical model. Many models have served chemists, ranging from the Bohr model through the most modern quantum theory; all have had adherents, detractors, uses, and flaws. When using such models, we must always avoid the trap of arguing that a particular model is *right* rather than arguing that it is *useful*. Models are just that - models.

Daylight's system represents molecules using a fairly standard valence model. For example, the Daylight system understands the normal valences of organic compounds, and by counting the bonding electrons in a

molecule, can fill in unspecified hydrogens, detect aromatic and anti-aromatic ring systems, and issue warnings when unlikely or impossible molecules are specified.

The Daylight system represents a molecule as a *graph* in which the *nodes* are atoms and the *edges* are bonds. Each atom has a several properties, including its atomic number, atomic weight, charge, and the number of attached hydrogens. If the atom is a chiral center, it can also have chiral specifications.

Bond properties are simpler: a bond is single, double, triple, or aromatic. The concept of aromaticity in the Daylight system is not a chemical one, but rather is a set of rules designed for a chemical nomenclature system (this is discussed more in the SMILES chapter).

There is some flexibility in this valence model. Molecules can be represented as a *hydrogen-suppressed* graph (hydrogen atoms are represented as a property of "heavy" atoms) or as a *hydrogen-complete* graph (hydrogens are represented the same way as other atoms). Bonds in cyclic structures can be represented as aromatic or as the alternating single/double bond Kekulé form. Isotopic information such as chirality and atomic mass can be unspecified, partially specified, or completely specified.

2.2 Analyzing Molecules

There are two classes of properties associated with a molecule and its constituent parts (atoms and bonds): *explicit properties* and *derived properties*. Explicit properties are those needed to completely specify the graph of a molecule: its atoms and bonds and their properties. Derived properties are properties that can be computed from the graph. The following sections describe the derived properties that are of interest in the Daylight system.

2.2.1 Cycles

When a molecule's atoms and bonds are specified, it may be that there are *cycles* in the graph. A bond is said to be a *ring bond* if it can be removed without breaking the graph (molecule) into two pieces. In this case, the graph is said to be *biconnected*). Atoms that are connected via ring bonds are *ring atoms*.

There are two parts to ring-detection in a graph:

- Ring detection: Discovering those atoms and bonds that are part of ring systems.
- Finding a smallest set of smallest rings (SSSR). In a multi-ring system there are many cyclic paths; for example a naphthalene system has two paths of length six around the two "obvious" rings plus a path of length ten around the perimeter. Any two of these three "rings" would completely describe the ring system, but the shortest cyclic paths are what one normally calls the "right" ones. Note that there is more than one valid SSSR for many systems - a tetrahedron has four equivalent faces but only three rings, making six valid SSSRs.

The algorithms used for these tasks are beyond the scope of this document. However, we will point out that of the two tasks above, the first is relatively trivial and the second is surprisingly complex. Many papers have been written describing algorithms for efficiently detecting SSSRs (G. Downs et al, *A Review of Ring Perception Algorithms for Chemical Graphs*, J. Chem. Inf. Comput. Sci. 29:172; 1989).

2.2.2 Bond Type, Bond Order, and Aromaticity

Bonds are, in a sense, both an explicit and a derived property. Although you generally specify the bond type of each bond, the Daylight system will sometimes rearrange them, such as in Kekulé ring systems.

The Daylight system defines *bond type* and *bond order* as follows:

Bond Order
: Bond order is one of single, double, or triple. Bond order is a *formal property*.

Bond Type
: A *derived property*; one of single, double, triple, or aromatic. The Daylight system uses an extended version of Hueckel's rule to identify aromatic molecules and ions. To qualify as aromatic, all atoms in a ring must be sp^2 hybridized and the number of available "excess" p-electrons must satisfy Hueckel's 4N+2 criterion. This is covered in more detail in the SMILES chapter.

Note that the definition of aromaticity is not intended to imply anything about the reactivity, magnetic resonance spectra, heat of formation, or odor of substances. Rather, the definition is designed to be useful in a chemical nomenclature system (SMILES) that is discussed in detail in the subsequent chapter.

2.2.3 Symmetry

A molecule's symmetry is useful for many purposes, including generating canonical labelings (such as when generating a *unique SMILES*), classifying chirality, detecting degenerate chiral specifications, and eliminating redundant calculations. The Daylight system automatically detects symmetry in the molecules it represents as those with two dimensional rotations.

2.2.4 Canonical Labeling

A computer representation of a molecule is often built in an arbitrary fashion; one can start with any atom on the molecule and add atoms and bonds in any order. If a "label" (typically a number) is assigned to each atom and bond as it is specified, the labeling is also arbitrary - a different input order of the same molecule results in a different set of labels.

Chemical nomenclature systems such as SMILES require a *canonical labeling* of the atoms and bonds - a numbering that is independent of the history of the molecule's representation. The Daylight system generates such a labeling whenever it generates a *unique SMILES*.

2.2.5 Chirality

Chirality, like bonds, is both an explicit property and an derived property. That is, the Daylight system accepts various chiral specifications on input, though it will sometimes change the specification to a different (but equivalent) form. Like the canonical labeling of atoms discussed above, a "canonical chiral representation" must be chosen if a unique SMILES is desired.

2.3 Representing Reactions

A reaction consists of an set of molecules, each of which plays a specific role in a reaction: reactant, product, or agent. Since reactions are made up of molecules, reactions naturally use the same valence model, bonding, aromaticity, and symmetry rules as molecules. At minimum, a reaction must contain valid molecules based on these rules.

In an ideal world (at least from an information-processing point of view), all reactions would be represented stoichiometrically (every relevant atom shown), and enough information would be present to tell unambiguously which atom was which between the reactants and products. This information would be provided by a pairwise mapping of the reactant and product atoms. In effect, the only difference between the reactant molecule(s) and product molecule(s) would be the bond changes and atom property changes

(chirality, charge, aromaticity) which occur during the reaction. If these criteria are met, one can 'superimpose' the reactants and products on one-another and represent the reaction as a reaction graph. This is both a complete and compact description of a reaction.

Unfortunately, these stringent requirements can rarely be met for reactions available in electronic form. The Daylight system is designed to be able to represent and store both completely specified (reaction graph-like) reactions and information-deficient reactions in a repeatable and searchable fashion. Although all of the molecules within a reaction must be chemically valid, an overall analysis of the reaction for chemical sensibility is not carried out.

The Daylight system is oriented towards single-step reactions with the following three roles for molecules defined:

Reactant
: A starting material. It is expected to participate in the reaction by contributing one or more atoms to the products. This participation is not enforced.

Agent
: Agents are molecules which do not contribute atoms to the product, or accept atoms from the reactants. Note that this definition of agents is manifested in the way atom maps are handled; they are ignored in the reaction object and are not part of the lexical definition of reactions (SMILES output). Agents are commonly used for catalysts, solvents and other adjuncts which participate indirectly in a reaction.

Product
: Molecules which are the final result of the reaction. All of the atoms in the product should come from the reactants. This is not enforced.

Note that the above distinctions between reactant, agent, and product all involve the participation of atoms in the reaction. This participation is recorded via the reaction atom map. The atom map simply maps the correspondence of the reactant and product atoms in the reaction. Agents never have meaningful atom maps, since by definition agent atoms do not participate directly in a reaction.

Clearly, reactions have additional data which one wants to store about them. The Daylight approach is to only encode the pure structural information in the lexical representation of the reaction and handle the additional data outside of the reaction. A standardized THOR database can allow coupling of the following data about the individual components of a reaction to those components:

- Stoichiometry
- Equilibrium constants
- Yield
- Conditions, (amounts, times, temperature, pressure, operations) which may or may not be expressed in a rigorous language

2.4 Depictions

One of the most important jobs of a chemical information system is to communicate information to the chemist effectively. One of the best ways to do this is graphically, using the standard schematic representation of chemicals familiar to all chemists.

The Daylight system provides an algorithm for generating these schematic diagrams *ab initio* - a drawing can be made of any molecule or reaction, whether or not it has ever been seen before. When generating a schematic diagram, two criteria are critical:

- Correctness - The schematic should correctly represent the molecule's graph
- Comprehensibility - The schematic should represent the atoms, bonds, and cycles in a sensible way, so that chemists will readily recognize familiar molecules and functional groups.
- Appearance - The aesthetic appearance is a subjective matter, but the Daylight system attempts to generate depictions that are pleasing to the eye. In some cases this is not possible to achieve in a reasonable time, in which case correctness and comprehensibility take precedence.

3. SMILES - A Simplified Chemical Language

SMILES (Simplified Molecular Input Line Entry System) is a *line notation* (a typographical method using printable characters) for entering and representing molecules and reactions. Some examples are:

SMILES	Name	SMILES	Name
CC	ethane	[OH3+]	hydronium ion
O=C=O	carbon dioxide	[2H]O[2H]	deuterium oxide
C#N	hydrogen cyanide	[235U]	uranium-235
CCN(CC)CC	triethylamine	F/C=C/F	E-difluoroethene
CC(=O)O	acetic acid	F/C=C\F	Z-difluoroethene
C1CCCCC1	cyclohexane	N[C@@H](C)C(=O)O	L-alanine
c1ccccc1	benzene	N[C@H](C)C(=O)O	D-alanine

Reaction SMILES	Name
[I-].[Na+].C=CCBr>>[Na+].[Br-].C=CCI	displacement reaction
(C(=O)O).(OCC)>>(C(=O)OCC).(O)	intermolecular esterification

SMILES contains the same information as might be found in an extended connection table. The primary reason SMILES is more useful than a connection table is that it is a linguistic construct, rather than a computer data structure. SMILES is a true language, albeit with a simple vocabulary (atom and bond symbols) and only a few grammar rules. SMILES representations of structure can in turn be used as "words" in the vocabulary of other languages designed for storage of chemical information (information about chemicals) and chemical intelligence (information about chemistry).

Part of the power of SMILES is that unique SMILES exist. With standard SMILES, the name of a molecule is synonymous with its structure; with unique SMILES, the name is universal. Anyone in the world who uses unique SMILES to name a molecule will choose the exact same name.

One other important property of SMILES is that it is quite compact compared to most other methods of representing structure. A typical SMILES will take 50% to 70% less space than an equivalent connection table, even binary connection tables. For example, a database of 23,137 structures, with an average of 20 atoms per structure, uses only 1.6 bytes per atom when represented with SMILES. In addition, ordinary compression of SMILES is extremely effective. The same database cited above was reduced to 27% of its original size by Ziv-Lempel compression (i.e. 0.42 bytes per atom).

These properties open many doors to the chemical information programmer. Examples of uses for SMILES are:

- Keys for database access
- Mechanism for researchers to exchange chemical information
- Entry system for chemical data
- Part of languages for artificial intelligence or expert systems in chemistry

The rest of this chapter is a concise exposition of the SMILES encoding rules. For further information, the reader is referred to *"SMILES 1. Introduction and Encoding Rules",* Weininger, D., *J.Chem. Inf. Comput. Sci.* 1988, 28,31.

3.1 Canonicalization

SMILES denotes a molecular structure as a graph with optional chiral indications. This is essentially the two-dimensional picture chemists draw to describe a molecule. SMILES describing only the labeled molecular graph (i.e. atoms and bonds, but no chiral or isotopic information) are known as generic SMILES. There are usually a large number of valid generic SMILES which represent a given structure. A canonicalization algorithm exists to generate one special generic SMILES among all valid possibilities; this special one is known as the "unique SMILES". SMILES written with isotopic and chiral specifications are collectively known as "isomeric SMILES". A unique isomeric SMILES is known as an "absolute SMILES". See the following examples.

Input SMILES	Unique SMILES
OCC	CCO
[CH3][CH2][OH]	CCO
C-C-O	CCO
C(O)C	CCO
OC(=O)C(Br)(Cl)N	NC(Cl)(Br)C(=O)O
ClC(Br)(N)C(=O)O	NC(Cl)(Br)C(=O)O
O=C(O)C(N)(Br)Cl	NC(Cl)(Br)C(=O)O

3.2 SMILES Specification Rules

SMILES notation consists of a series of characters containing no spaces. Hydrogen atoms may be omitted (hydrogen-suppressed graphs) or included (hydrogen-complete graphs). Aromatic structures may be specified directly or in Kekulé form.

There are five generic SMILES encoding rules, corresponding to specification of atoms, bonds, branches, ring closures, and disconnections. Rules for specifying various kinds of isomerism are discussed in the following section, *ISOMERIC SMILES*.

3.2.1 Atoms

Atoms are represented by their atomic symbols: this is the only required use of letters in SMILES. Each non-hydrogen atom is specified independently by its atomic symbol enclosed in square brackets, **[ ]**. The second letter of two-character symbols must be entered in lower case. Elements in the "organic subset" **B, C, N, O, P, S, F, Cl, Br,** and **I** may be written without brackets if the *number of attached hydrogens conforms to the lowest normal valence consistent with explicit bonds*. "Lowest normal valences" are B (3), C (4), N (3,5), O (2), P (3,5), S (2,4,6), and 1 for the halogens. Atoms in aromatic rings are specified by lower case letters, e.g., aliphatic carbon is represented by the capital letter **C**, aromatic carbon by lower case **c**. Since attached hydrogens are implied in the absence of brackets, the following atomic symbols are valid SMILES notations.

C	methane	(CH4)
P	phosphine	(PH3)
N	ammonia	(NH3)
S	hydrogen sulfide	(H2S)
O	water	(H2O)
Cl	hydrochloric acid	(HCl)

Atoms with valences other than "normal" and elements not in the "organic subset" must be described in brackets.

[**S**]	elemental sulfur
[**Au**]	elemental gold

Within brackets, any attached hydrogens and formal charges must always be specified. The number of attached hydrogens is shown by the symbol **H** followed by an optional digit. Similarly, a formal charge is shown by one of the symbols **+** or **-**, followed by an optional digit. If unspecified, the number of attached hydrogens and charge are assumed to be zero for an atom inside brackets. Constructions of the form [Fe+++] are synonymous with the form [Fe+3]. Examples are:

[**H+**]	proton
[**Fe+2**]	iron (II) cation
[**OH-**]	hydroxyl anion
[**Fe++**]	iron (II) cation
[**OH3+**]	hydronium cation
[**NH4+**]	ammonium cation

3.2.2 Bonds

Single, double, triple, and aromatic bonds are represented by the symbols -, =, #, and :, respectively. Adjacent atoms are assumed to be connected to each other by a single or aromatic bond (single and aromatic bonds may always be omitted). Examples are:

CC	ethane	(CH3CH3)
C=O	formaldehyde	(CH2O)
C=C	ethene	(CH2=CH2)
O=C=O	carbon dioxide	(CO2)
COC	dimethyl ether	(CH3OCH3)
C#N	hydrogen cyanide	(HCN)
CCO	ethanol	(CH3CH2OH)
[H][H]	molecular hydrogen	(H2)

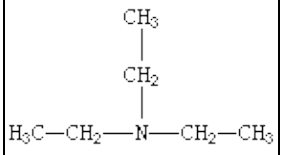
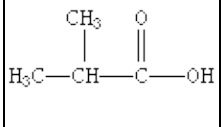
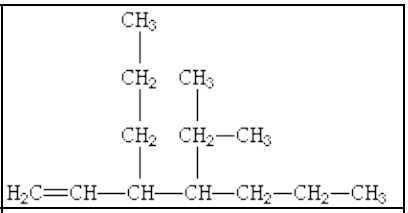
For linear structures, SMILES notation corresponds to conventional diagrammatic notation except that hydrogens and single bonds are generally omitted. For example, 6-hydroxy-1,4-hexadiene can be represented by many equally valid SMILES, including the following three:

Structure	**Valid SMILES**

	C=CCC=CCO
CH2=CH-CH2-CH=CH-CH2-OH	C=C-C-C=C-C-O
	OCC=CCC=C

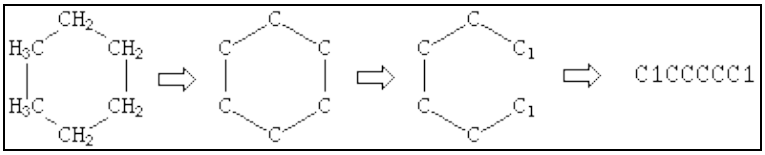
3.2.3 Branches

Branches are specified by enclosing them in parentheses, and can be nested or stacked. In all cases, the implicit connection to a parenthesized expression (a "branch") is to the left. Examples are:

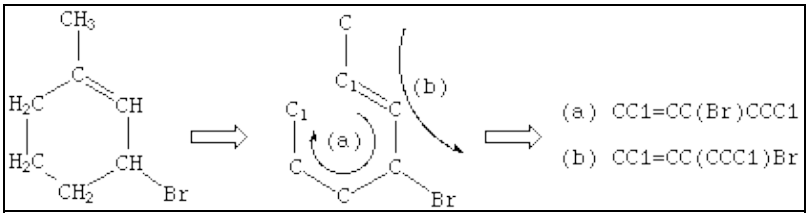
		
CCN(CC)CC	CC(C)C(=O)O	C=CC(CCC)C(C(C)C)CCC
Triethylamine	**Isobutyric acid**	**3-propyl-4-isopropyl-1-heptene**

3.2.4 Cyclic Structures

Cyclic structures are represented by breaking one bond in each ring. The bonds are numbered in any order, designating ring opening (or ring closure) bonds by a digit immediately following the atomic symbol at each ring closure. This leaves a connected non-cyclic graph which is written as a non-cyclic structure using the three rules described above. Cyclohexane is a typical example:

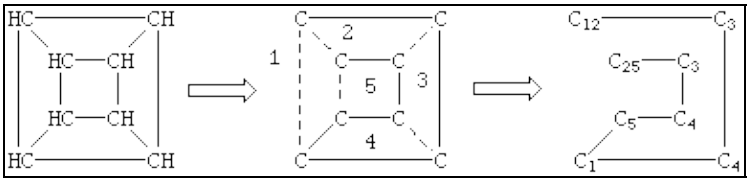


There are usually many different, but equally valid descriptions of the same structure, e.g., the following SMILES notations for 1-methyl-3-bromo-cyclohexene-1:



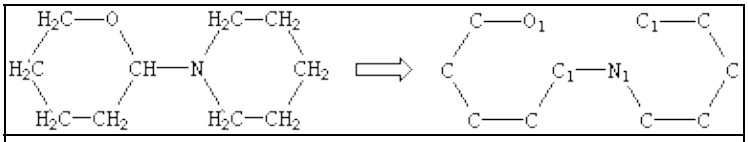
Many other notations may be written for the same structure, deriving from different ring closures. SMILES does not have a preferred entry on input; although (a) above may be simplest, others are just as valid.

A single atom may have more than one ring closure. This is illustrated by the structure of cubane in which two atoms have more than one ring closure:



Generation of SMILES for cubane: **C12C3C4C1C5C4C3C25**.

If desired, digits denoting ring closures can be reused. As an example, the digit 1 used twice in the specification:

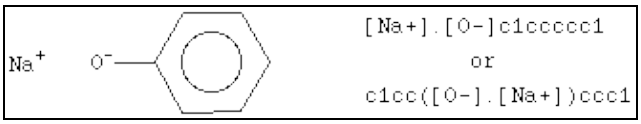


O1CCCCC1N1CCCCC1

The ability to re-use ring closure digits makes it possible to specify structures with 10 or more rings. Structures that require more than 10 ring closures to be open at once are exceedingly rare. If necessary or desired, higher-numbered ring closures may be specified by prefacing a two-digit number with percent sign (**%**). For example, **C2%13%24** is a carbon atom with a ring closures 2, 13, and 24 .

3.2.5 Disconnected Structures

Disconnected compounds are written as individual structures separated by a "**.**" (period). The order in which ions or ligands are listed is arbitrary. There is no implied pairing of one charge with another, nor is it necessary to have a net zero charge. If desired, the SMILES of one ion may be imbedded within another as shown in the example of sodium phenoxide.



Matching pairs of digits following atom specifications imply that the atoms are bonded to each other. The bond may be explicit (bond symbol and/or direction preceding the ring closure digit) or implicit (a nondirectional single or aromatic bond). This is true whether or not the bond ends up as part of a ring.

Adjacent atoms separated by dot (.) implies that the atoms are not bonded to each other. This is true whether or not the atoms are in the same connected component.

For example, **C1.C1** specifies the same molecule as **CC**(ethane)

3.3 Isomeric SMILES

This section describes the SMILES rules used to specify isotopism, configuration about double bonds, and chirality. The term *isomeric SMILES* collectively refers to SMILES written using these rules.

The SMILES isomer specification rules allow chirality to be completely specified for any structure, if it is known. Unlike most existing chemical nomenclatures such as CIP and IUPAC, these rules are also designed to allow rigorous *partial* specification of chirality. Aside from use in macros, substructure searching, and other pattern matching operations, this is important because much of the world's available chemical

information is known for structures with incompletely resolved chiralities (not all possible chiral centers are separated, known, or reported).

All isomer specification rules in SMILES are therefore optional. The absence of a specification for any attribute implies that the value of that attribute is unspecified.

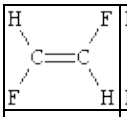
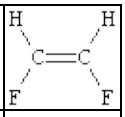
3.3.1 Isotopic Specification

Isotopic specifications are indicated by preceding the atomic symbol with a number equal to the desired integral atomic mass. An atomic mass can only be specified inside brackets. For instance:

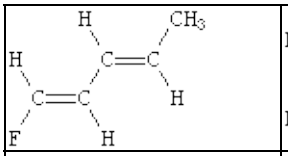
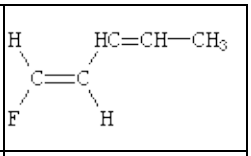
Smiles	Name
[**12C**]	carbon-12
[**13C**]	carbon-13
[**C**]	carbon (unspecified mass)
[**13CH4**]	C-13 methane

3.3.2 Configuration Around Double Bonds

Configuration around double bonds is specified by the characters **/** and **** which are "directional bonds" and can be thought of as kinds of single or aromatic (eg. default) bonds. These symbols indicate relative directionality between the connected atoms, and have meaning only when they occur on both atoms which are double bonded. For instance, the following SMILES are all valid for E- and Z-1,2-difluoroethene:

	
F/C=C/F	F/C=C\F
F\C=C\F	F\C=C/F

An important difference between SMILES chirality conventions and others such as CIP is that SMILES uses local chirality representation (as opposed to absolute chirality), which allows *partial* specifications. An example of this is illustrated below:

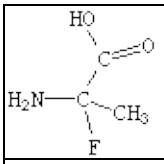
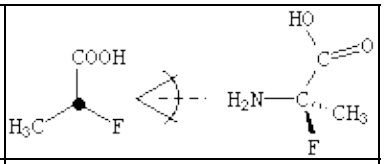
	
F/C=C/C=C/C	F/C=C/C=CC
(completely specified)	(partially specified)

3.3.3. Configuration Around Tetrahedral Centers

SMILES uses a very general type of chirality specification based on local chirality. Instead of using a rule-based numbering scheme to order neighbor atoms of a chiral center, orientations are based on the order in which neighbors occur in the SMILES string. As with all other aspects of SMILES, any valid order is acceptable; the Daylight software is responsible for retaining the meaning of the chiral specification when the structure is modified or rearranged (e.g. to make the unique SMILES).

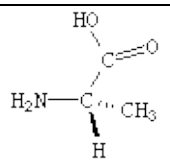
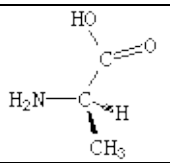
The simplest and most common kind of chirality is tetrahedral; four neighbor atoms are evenly arranged about a central atom, known as the "chiral center". If all four neighbors are different from each other in any way, mirror images of the structure will not be identical. The two mirror images are known as "enantiomers" and are the only two forms that a tetrahedral center can have. If two (or more) of the four neighbors are identical to each other, the central atom will not be chiral (its mirror images can be superimposed in space).

In SMILES, tetrahedral centers may be indicated by a simplified chiral specification (`@` or `@@`) written as an atomic property following the atomic symbol of the chiral atom. If a chiral specification is not present for a chiral atom, its chirality is implicitly not specified. For instance:

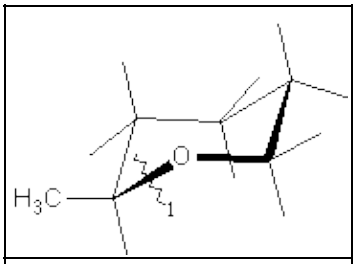
	
NC(C)(F)C(=O)O	N[C@](C)(F)C(=O)O
NC(F)(C)C(=O)O	N[C@@](F)(C)C(=O)O
(unspecified chirality)	(specified chirality)

Looking from the amino `N` to the chiral `C` (as the SMILES is written), the three other neighbors appear anticlockwise in the order that they are written in the top SMILES, `N[C@](C)(F)C(=O)O` (methyl-C, F, carboxy-C), and clockwise in the bottom one, `N[C@@](F)(C)C(=O)O`. The symbol "`@`" indicates that the following neighbors are listed anticlockwise (it is a "visual mnemonic" in that the symbol looks like an anticlockwise spiral around a central circle). "`@@`" indicates that the neighbors are listed clockwise (you guessed it, anti-anti-clockwise).

If the central carbon is not the very first atom in the SMILES and has an implicit hydrogen attached (it can have at most one and still be chiral), the implicit hydrogen is taken to be the first neighbor atom of the three neighbors that follow a tetrahedral specification. If the central carbon is first in the SMILES, the implicit hydrogen is taken to be the "from" atom. Hydrogens may always be written explicitly (as `[H]`) in which case they are treated like any other atom. In each case, the *implied order is exactly as written in SMILES*. Some of the valid SMILES for the alanine are:

	
N[C@@](C)(C(=O)O)[H]	N[C@](C)(C(=O)O)[H]
N[C@@H](C(=O)O)C	N[C@H](C(=O)O)C
N[C@H](C)C(=O)O	N[C@@H](C)C(=O)O
[H][C@](C)(N)C(=O)O	[H][C@@](C)(N)C(=O)O
[C@H](C)(N)C(=O)O	[C@@H](C)(N)C(=O)O

The chiral order of the ring closure bond is implied by the lexical order that the ring closure digit appears on the chiral atom (not in the lexical order of the "substituent" atom).



C[C@H]1CCCCO1
or
O1CCCC[C@@H]1C

3.3.4 General Chiral Specification

There are many kinds of chirality other than tetrahedral. The use of the "@" symbol described above is actually a special case of a general chiral specification syntax.

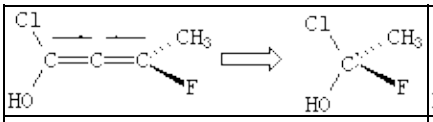
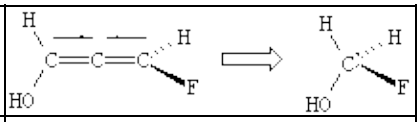
The general chiral specification used in SMILES has three parts: the @ symbol, followed by a two-letter chiral class indicator, followed by a numerical chiral permutation designator. A default chiral class is assigned to each degree (number of connections); the default class for four connections is tetrahedral (TH). Most chiralities have more than two possible choices; the choices are assigned from a table numerically. In most cases, the @1 designation means "anticlockwise around the axis represented by SMILES order" and @2 means "clockwise". Notations in the form "@@" and "@@@" are interpreted as "@2" and "@3" (analogous to "+++" meaning "+3"). The "@" and "@@" notations used above are shortcuts for the full specifications "@TH1" and "@TH2". In practice, full chiral specifications are not often needed.

SMILES handles the full range of chiral specification, including resolution of "reduced chirality" (where the number of enantiomers is reduced by symmetry) and "degenerate chirality" (where the center becomes non-chiral due to symmetrical substitution). As with other aspects of SMILES, the language guarantees the ability to specify exactly what is known, including partial specifications. The SMILES system will generate unique isomeric SMILES for any given specification, and substructure recognition will operate correctly on all types of chirality.

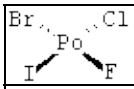
The rest of this section will be limited to discussing the following chiralities: tetrahedral, allene-like, square-planar, trigonal-bipyramidal, and octahedral. Although many more chiral classes can be handled by this system (it's table-driven), these five classes are very common in chemistry and cover most of the issues to be encountered in the remainder.

Tetrahedral. The tetrahedral class symbol is TH. This is the default chiral class for degree four. Possible values are 1 and 2. @TH1 (or just @) indicates that, looking from the first connected atom, the following three connected atoms are listed anticlockwise; @TH2 (or @@) indicates clockwise.

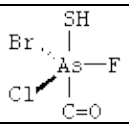
Allene-like. The allene-like class symbol is AL. This is the default chiral class for degree 2 (the chiral center is the central atom with two double bonds). Although substituted C=C=C structures are most common, C=C=C=C=C structures are also allene-like, as are any odd number of serially double-bonded atoms. Possible values are @AL1 (or just @) and @AL2 (or @@); these are interpreted by superimposing the substituted atoms and evaluating as per tetrahedral. Hydrogens attached to substituted allene-like atoms are taken to be immediately following that atom, as shown below:

	
OC(Cl)=[C@]=C(C)F	OC=[C@]=CF
OC(Cl)=[C@AL1]=C(C)F	OC([H])=[C@AL1]=C([H])F

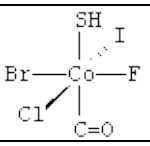
Square-planar. The square-planar class symbol is **SP** Possible values are `@SP1`, `@SP2`, and `@SP3`; this is not the default chiral class for degree four, so shorthand specifications are not allowed. Square-planar is also somewhat unusual in that the ideas of clockwise and anticlockwise do not apply.

	
F[Po@SP1](Cl)(Br)I	(SP1 lists in a "U shape")
F[Po@SP2](Br)(Cl)I	(SP2 lists in a "4-shape")
F[Po@SP3](Cl)(I)Br	(SP3 lists in a "Z shape")

Trigonal-bipyramidal. The trigonal-bipyramidal class symbol is TB. This is the default chiral class for degree five. Possible values are `@TB1` to `@TB20`. `@TB1` (or just `@`) indicates that, when the SMILES is listed from one axial connection to the other, the three intermediate, equatorially-connected atoms are listed anticlockwise; `@TB2` (or `@@`) indicates clockwise. This is illustrated below.


s[As@@](F)(Cl)(Br)C=O
O=C[As@](F)(Cl)(Br)S

Octahedral. The octahedral class symbol is `OH`. This is the default chiral class for degree six. Possible values are `@OH1` to `@OH30`. `@OH1` (or just `@`) indicates that, when the SMILES is listed from one axial connection to the other, the four intermediate, equatorially-connected atoms are listed anticlockwise; `@OH2` (or `@@`) indicates clockwise. This is illustrated below.


S[Co@@](F)(Cl)(Br)(I)C=O
O=C[Co@](F)(Cl)(Br)(I)S

3.4 SMILES Conventions

Aside from the above rules, a small number of conventions are universally used in SMILES. These are briefly discussed below; for more detail, see the JCICS paper (*ibid*).

3.4.1 Hydrogens

Hydrogen atoms do not normally need to be specified when writing SMILES for most organic structures. The presence of hydrogens may be specified in three ways:

- Implicitly.....for atoms specified without brackets, from normal valence assumptions.
- Explicitly by count.....inside brackets, by the hydrogen count supplied; zero if unspecified.
- As explicit atoms.....as [H] atoms.

There is no distinction between "organic" and "inorganic" SMILES nomenclature. One may specify the number of attached hydrogens for any atom in any SMILES. For example, propane may be entered as `[CH3][CH2][CH3]` instead of `CCC`.

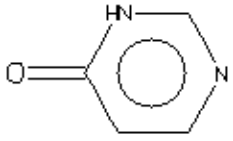
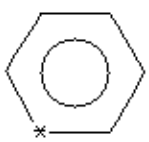
There are four situations where specification of explicit hydrogen specification is required:

- charged hydrogen, i.e. a proton, `[H+]`;
- hydrogens connected to other hydrogens, e.g., molecular hydrogen, `[H][H]`;
- hydrogens connected to other than one other atom, e.g., bridging hydrogens; and
- isotopic hydrogen specifications, e.g. in heavy water, `[2H]O[2H]`.

3.4.2 Aromaticity

Aromaticity must be deduced in a system such as SMILES which generates an unambiguous chemical nomenclature because of the fundamental requirement to characterize the symmetry of a molecule. Given effective aromaticity-detection algorithms, it is not necessary to enter any structure as aromatic if the user prefers to enter an aliphatic (Kekulé-like) structure. Entering structures as aromatic directly (i.e., by using lower case atomic symbols) provides a shortcut to accurate chemical specification and is closer to the mental molecular model used by most chemists.

The SMILES algorithm uses an extended version of Hueckel's rule to identify aromatic molecules and ions. To qualify as aromatic, all atoms in the ring must be sp^2 hybridized and the number of available "excess" p-electrons must satisfy Hueckel's 4N+2 criterion. As an example, benzene is written `c1ccccc1`, but an entry of `C1=CC=CC=C1` - cyclohexatriene, the Kekulé form - leads to detection of aromaticity and results in an internal structural conversion to aromatic representation. Conversely, entries of `c1ccc1` and `c1cccccccc1` will produce the correct anti-aromatic structures for cyclobutadiene and cyclooctatetraene, `C1=CC=C1` and `C1=CC=CC=CC=C1`. In such cases the SMILES system looks for a structure that preserves the implied sp^2 hybridization, the implied hydrogen count, and the specified formal charge, if any. Some inputs, however, may not only be incorrect but also impossible, such as `c1cccc1`. Here `c1cccc1` cannot be converted to `C1=CCC=C1` since one of the carbon atoms would be sp^3 with two attached hydrogens. In such a structure alternating single and double bond assignments cannot be made. The SMILES system will flag this as an "impossible" input. Please note that only atoms on the following list can be considered aromatic: C, N, O, P, S, As, Se, and * (wildcard). In addition, exocyclic double bonds do not break aromaticity.

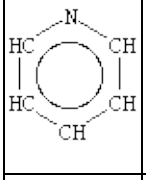
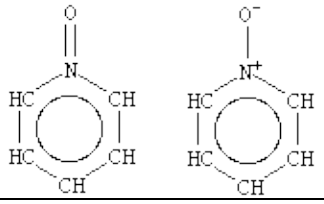
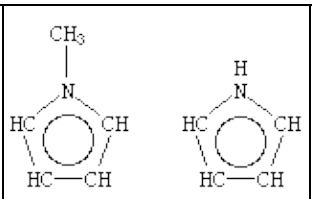
		
C1=COC=C1	C1=CN=C[NH]C(=O)1	C1=C*=CC=C1
c1cocc1	c1cnc[nH]c(=O)1	c1c*ccc1

It is important to remember that the purpose of the SMILES aromaticity detection algorithm is for the purposes of chemical information representation only! To this end, rigorous rules are provided for determining the "aromaticity" of charged, heterocyclic, and electron-deficient ring systems. The "aromaticity" designation as used here is not intended to imply anything about the reactivity, magnetic resonance spectra,

heat of formation, or odor of substances.

3.4.3 Aromatic Nitrogen Compounds

A short note is in order about aromatic nitrogens, a common source of confusion in chemical information systems. All three common types of aromatic nitrogen may be specified with the aromatic nitrogen symbol **n**. Archetypical examples are pyridine, pyridine-N-oxide, and pyrrole.

		
n1ccccc1	O=n1ccccc1 [O-][n+]1ccccc1	Cn1cccc1 [nH]1cccc1
Pyridine	**Pyridine-N-oxide**	**Methyl and 1H-pyrrole**

Note that the pyrrolyl nitrogen in 1H-pyrrole is written [nH] to distinguish this kind of nitrogen from a pyridyl-N. Alternative valid SMILES for 1H-pyrrole include `[H]n1cccc1` (with explicit hydrogen) and `N1C=CC=C1` (aliphatic form) all three input forms are equivalent.

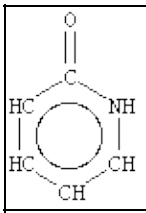
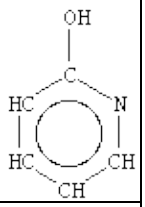
3.4.4 Bonding Conventions

SMILES does not dictate which valence conventions should be used to model molecular structure. In fact, an advantage of using SMILES is its ability to describe various valence models of the same structure. Atoms may be connected and show charge separation as desired. For instance, nitromethane can be represented in SMILES as `CN(=O)=O` or as the charge separated `C[N+](=O)[O-]` (we tend to use the former for database work because it preserves symmetry). Both are "right" in the sense that they represent different, useful models of the substance. In general, when symmetry is not an issue, most chemists prefer charge-separated structures if they can avoid representing atoms in unusual valence states, e.g., diazomethane is written as `C=[N+]=[N-]` rather than `C=[N]=[N]`.

Given one valence model of a structure, chemical database systems such as THOR and Merlin have the ability to retrieve data about that structure even if the data were stored under a different valence model of the structure. With such systems, the choice of valence conventions is not critical to either database design nor database query.

3.4.5 Tautomers

Tautomeric structures are explicitly specified in SMILES. There are no "tautomeric bond", "mobile hydrogen", nor "mobile charge" specifications. Selection of one or all tautomeric structures is left to the user and strongly depends on the application. Given one tautomeric form, most chemical information systems will report data for all known tautomers if needed. The role of SMILES is to specify exactly which tautomeric form is requested, and for which there are data. A simple example, with two possible tautomeric forms, is shown below:

	
O=c1[nH]cccc1	Oc1ncccc1
2-pyridone	**2-pyridinol**

3.5 Extensions for Reactions

The SMILES language is extended to handle reactions. There are two areas where SMILES is extended: distinguishing component parts of a reactions and atom maps.

Component parts of a reaction are handled by introducing the ">" character as a new separator. Any reaction must have exactly two > characters in it. ">>" is a valid reaction SMILES for an empty reaction. Each of the ">"-separated components of a reaction must be a valid molecule SMILES.

As an aside, molecule SMILES never have a ">" character. In a program, one can quickly determine if a SMILES refers to a reaction or molecule by searching for a ">" character in the string.

Reaction SMILES Grammar:

```
reaction                :       reactant '>' agent '>' product
                        |       reactant '>>' product
                        ;
reactant,
agent,
product                 :       molecule
                        |       <null>
                        ;
molecule                :       SMILES
                        ;
```

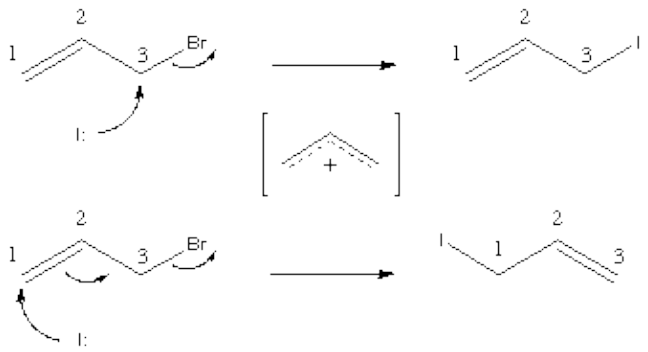
SMILES: a valid molecule specification in the SMILES language. For example:

C=CCBr>>C=CCI
: This is a valid reaction. Note that there are no agent molecules. Also note that several atoms are missing from the reaction (the product "Br" and the reactant "F").

[I-].[Na+].C=CCBr>>[Na+].[Br-].C=CCI
: This is a more complete version of the same reaction. It has been canonicalized. It would form the root of a datatree when stored in a THOR database.

C=CCBr.[Na+].[I-]>CC(=O)C>C=CCI.[Na+].[Br-]
: This version of the reaction includes an agent. Note that the SMILES does not indicate how the agent participates. Whether the agent is a solvent, catalyst, or performs another function within the reaction must be stored separately as data. This SMILES could be stored in a THOR database as an absolute SMILES and would appear on the same datatree page as the previous example.

In the above example, note that the reaction is ambiguous with respect to the carbon atoms involved. One might assume that a normal Sn2 displacement is occurring. In fact, an equally reasonable allylic displacement is possible, via either an Sn1-like allyl cation. Recognize that the reaction SMILES given above do not say which carbons are which and hence do not discriminate between the two alternate mechanisms.



This case demonstrates the use and need for atom maps for reaction processing. Atom maps are used primarily to further define the overall reaction in cases where the reaction mechanism may not be evident from the reactant and product molecules. Atom maps are non-negative integer atom modifiers. They follow the ":" character within an atom expression. They must be the last modifier within the atom expression:

SMILES Atom Expression Grammar:

```
atom                    :       SYMBOL
                        |       [ WEIGHT SYMBOL mods ]
                        |       [ WEIGHT SYMBOL mods : CLASS ]
                        ;
mods                    :       mod mods
                        ;
mod                     :       HCOUNT | CHARGE | CHIRAL
                        ;
```

CLASS = non-negative integer class value. WEIGHT = atomic weight. SYMBOL = atomic symbol. HCOUNT = Atom hydrogen count specification. CHARGE = Atom charge specification. CHIRAL = Atom chirality specification.

Atom maps are an *atomic* property. They can legally appear in a SMILES for any atom, whether or not it is part of a reaction. Atom with atom map labels in a molecule SMILES are considered valid; the atom maps are ignored for molecule processing. Absolute and unique SMILES generated by the system for molecules never include atom maps.

Finally, there are some differences in the handling of atom maps and agent components in the *unique* versus *absolute* SMILES for reactions. Atom maps and agent components are not part of the unique SMILES specification. This is important for the THOR database, where the datatree roots are formed from the unique SMILES. The net result is that each reaction datatree may contain multiple specific reactions with different agents and atom maps.

3.5.1 Reaction Atom Maps

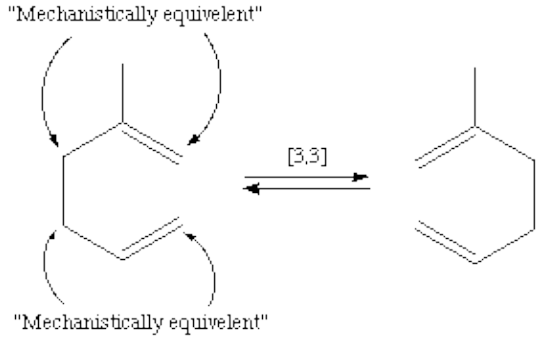
Atom mappings are properties of the atoms in the reaction molecules. The mappings represent equivalence classes of atoms within a reaction. In effect, the map tells the computer which atoms are the same on the reactant and products sides of a reaction. Without this map information, it is difficult to derive the reaction bond changes which occur.

Within the SMILES language, atom maps are represented as a non-negative numeric atom modifier following the ":" character (e.g. [CH3:2] is a carbon in class 2).

Within the Daylight toolkit, the atom maps are manipulated as sets of mapped atoms. The atom map class numbers which are used in SMILES do not appear in the toolkit interface to a reaction. The map class numbers in SMILES do not have any additional significance, except to associate all atoms with the same map class label to one another.

There are no requirements for completeness or uniqueness of the atom mappings. Atom mappings are independent of the connectivity and properties of the underlying molecules. This is so for several reasons: first, there are limits to the valence representation of molecules which appear when processing reactions. For example the oxygens in sodium acetate (CC(=O)[O-].[Na+]) are chemically indistinguishable, even though the valence model used in the toolkit requires that they be connected differently. Some systems (CAS, for example) recognize this equivalence in their structural representation (the tautomer bond). It is often useful to map these to the same class for reaction purposes: **[CH3:1][C:2](=[O:3])[O-:3].[Na+:4]**

A second case is where there is ambiguity in a reaction mechanism which one wants to express:



can undergo a cope rearrangement before reaction (which yields the same molecule graph). In effect, there are two distinct mechanisms by which the product is produced. This can be expressed as part of a reaction by: **[CH2:1]=[CH:2][CH2:1][CH2:3][C:4](C)[CH2:3]**

A third case is simply a lack of information about the reaction itself. It should be possible to omit some atom maps or specify partial information for sets of atoms which *might* end up in a given position in the product. It is never acceptable to force a user to make up data in order to register a reaction. One should only store exactly what is known about the reaction. Atom maps are, by definition ambiguous with respect to the underlying molecules. Atom maps do not appear in the lexical representation of a unique SMILES. They do appear in the lexical representation of an absolute SMILES.

Finally, atom maps are arbitrary class designations; the values of the numbers have no meaning. The Daylight system reserves the right to change the class numbers upon canonicalization of a reaction. The system will reorder the atom map classes over the entire reaction during canonicalization. The resulting maps are guaranteed to have the same meaning as the reaction before canonicalization. Practically, the maps are renumbered as small, dense integers in canonical atom order, but this is not guaranteed. Also, during canonicalization, the atom map classes for agent atoms are removed.

3.5.2 Hydrogens

Hydrogens in reactions are handled as with molecules; they are suppressed unless "special". Recall that for molecules, hydrogens are special if they are: charged, isotopic, bonded to another hydrogen, or multiply bonded. With reactions, there is an additional case which will make a hydrogen special. It is often desirable (eg. 1,5-hydride shift) to store information about the location of hydrogens as part of the atom map of a reaction. Hydrogens with a supplied atom map are considered "special" and these hydrogens are not

suppressed. These mapped hydrogens appear explicitly in Absolute SMILES for reactions. Otherwise, atom-mapped hydrogens do not appear in Unique SMILES.

3.6 Acknowledgments

Development of SMILES was initiated by the author, David Weininger, at the Environmental Research Laboratory, U.S.E.P.A., Duluth, MN; the design was completed at Pomona College in Claremont, CA. It was embodied in the Daylight Toolkit with the assistance of Cedar River Software.

4. SMARTS - A Language for Describing Molecular Patterns

Substructure searching, the process of finding a particular pattern (subgraph) in a molecule (graph), is one of the most important tasks for computers in chemistry. It is used in virtually every application that employs a digital representation of a molecule, including depiction (to highlight a particular functional group), drug design (searching a database for similar structures and activity), analytical chemistry (looking for previously-characterized structures and comparing their data to that of an unknown), and a host of other problems.

SMARTS is a language that allows you to specify substructures using rules that are straightforward extensions of SMILES. For example, to search a database for phenol-containing structures, one would use the SMARTS string **[OH]c1ccccc1**, which should be familiar to those acquainted with SMILES. In fact, almost all SMILES specifications are valid SMARTS targets. Using SMARTS, flexible and efficient substructure-search specifications can be made in terms that are meaningful to chemists.

In the SMILES language, there are two fundamental types of symbols: *atoms* and *bonds*. Using these SMILES symbols, once can specify a molecule's graph (its "nodes" and "edges") and assign "labels" to the components of the graph (that is, say what type of atom each node represents, and what type of bond each edge represents).

The same is true in SMARTS: One uses atomic and bond symbols to specify a graph. However, in SMARTS the labels for the graph's nodes and edges (its "atoms" and "bonds") are extended to include "logical operators" and special atomic and bond symbols; these allow SMARTS atoms and bonds to be more general. For example, the SMARTS atomic symbol **[C,N]** is an atom that can be aliphatic **C** or aliphatic **N**; the SMARTS bond symbol **~** (tilde) matches any bond.

4.1 Atomic Primitives

SMARTS provides a number of primitive symbols describing atomic properties beyond those used in SMILES (atomic symbol, charge, and isotopic specifications). The following tables list the atomic primitives used in SMARTS (all SMILES atomic symbols are also legal). In these tables **<n>** stands for a digit, **<c>** for chiral class.

Note that atomic primitive **H** can have two meanings, implying a property or the element itself. **[H]** means hydrogen atom. **[*H2]** means any atom with exactly two hydrogens attached

SMARTS Atomic Primitives

Symbol	Symbol name	Atomic property requirements	Default
*	wildcard	any atom	(no default)

a	aromatic	aromatic	(no default)
A	aliphatic	aliphatic	(no default)
D<n>	degree	<n> explicit connections	exactly one
H<n>	total-H-count	<n> attached hydrogens	exactly one[1]
h<n>	implicit-H-count	<n> implicit hydrogens	at least one
R<n>	ring membership	in <n> SSSR rings	any ring atom
r<n>	ring size	in smallest SSSR ring of size <n>	any ring atom[2]
v<n>	valence	total bond order <n>	exactly one[2]
X<n>	connectivity	<n> total connections	exactly one[2]
x<n>	ring connectivity	<n> total ring connections	at least one[2]
- <n>	negative charge	-<n> charge	-1 charge (-- is -2, etc)
+<n>	positive charge	+<n> formal charge	+1 charge (++ is +2, etc)
#n	atomic number	atomic number <n>	(no default)[2]
@	chirality	anticlockwise	anticlockwise, default class[2]
@@	chirality	clockwise	clockwise, default class[2]
@<c><n>	chirality	chiral class <c> chirality <n>	(nodefault)
@<c><n>?	chiral or unspec	chirality <c><n> or unspecified	(no default)
<n>	atomic mass	explicit atomic mass	unspecified mass

[1] Semantics of [H] changed in v4.5 [2] @ and @@ introduced in v4.1; r, v, X, and # in v4.3; x in v4.9

Caliphatic carbon atomcaromatic carbon atomaaromatic atom[#6]carbon atom[Ca]calcium atom[++]atom with a +2 charge[R]atom in any ring[D3]atom with 3 explicit bonds (implicit H's don't count)[X3]atom with 3 total bonds (includes implicit H's)[v3]atom with bond orders totaling 3 (includes implicit H's)C[C@H](F)Omatch chirality (H-F-O anticlockwise viewed from C)C[C@?H](F)Omatches if chirality is as specified or is not specified

4.2 Bond Primitives

Various bond symbols are available to match connections between atoms. A missing bond symbol is interpreted as "single or aromatic".

SMARTS Bond Primitives

SymbolAtomic property requirements-single bond (aliphatic)/directional bond "up"[1]\directional bond "down"[1]/?directional bond "up or unspecified"\?directional bond "down or unspecified"=double bond#triple bond:aromatic bond~any bond (wildcard)@any ring bond[1]

[1]/ and \ introduced in v4.1; @ in v4.6

Examples:

Cany aliphatic carbonccany pair of attached aromatic carbonsc:caromatic carbons joined by an aromatic bondc-caromatic carbons joined by a single bond (e.g. biphenyl)

4.3 Logical Operators

Atom and bond primitive specifications may be combined to form expressions by using logical operators. In the following table, **e** is an atom or bond SMARTS expression (which may be a primitive). The logical operators are listed in order of decreasing precedence (high precedence operators are evaluated first).

SMARTS Logical Operators

Symbol **Expression** **Meaning**
exclamation !e1 not e1
ampersand e1&e2 a1 and e2 (high precedence)
comma e1,e2 e1 or e2
semicolon e1;e2 a1 and e2 (low precedence)

All atomic expressions which are not simple primitives must be enclosed in brackets. The default operation is **&** (high precedence "and"), i.e., two adjacent primitives without an intervening logical operator must both be true for the expression (or subexpression) to be true.

The ability to form expressions gives the SMARTS user a great deal of power to specify exactly what is desired. The two forms of the AND operator are used in SMARTS instead of grouping operators.

Examples:

[CH2] aliphatic carbon with two hydrogens (methylene carbon)
[!C;R] (NOT aliphatic carbon) AND in ring
[!C;!R0] same as above ("!R0" means not in zero rings)
[n;H1] H-pyrrole nitrogen
[n&H1] same as above
[nH1] same as above
[c,n&H1] any arom carbon OR H-pyrrole nitrogen
[X3&H0] atom with 3 total bonds and no H's
[c,n;H1] (arom carbon OR arom nitrogen) and exactly one H
[Cl] any chlorine atom
[35*] any atom of mass 35
[35Cl] chlorine atom of mass 35
[F,Cl,Br,I] the 1st four halogens.

4.4 Recursive SMARTS

Any SMARTS expression may be used to define an atomic environment by writing a SMARTS starting with the atom of interest in this form:

$(SMARTS)

Such definitions may be considered atomic properties. These expressions can be used in same manner as other atomic primitives (also, they can be nested). Recursive SMARTS expressions are used in the following manner:

*C	atom connected to methyl (or methylene) carbon
*CC	atom connected to ethyl carbon
[$(*C);$(*CC)]	atom in both above environments (matches CCC)

The additional power of such expressions is illustrated by the following example which derives an expression for methyl carbons which are ortho to oxygen and meta to a nitrogen on an aromatic ring.

CaaO	C ortho to O
CaaaN	C meta to N
Caa(O)aN	C ortho to O and meta to N (but 2O,3N only)
Ca(aO)aaN	C ortho to O and meta to N (but 2O,5N only)
C[$(aaO);$(aaaN)]	C ortho to O and meta to N (all cases)

4.5 Component-level grouping of SMARTS

SMARTS may contain "zero-level" parentheses which can be used to group dot-disconnected fragments. This grouping operator allows SMARTS to express more powerful component queries. In general, a single set of parentheses may surround any legal SMARTS expression. Two or more of these expressions may be combined into more complex SMARTS:

(SMARTS)
(SMARTS).(SMARTS)
(SMARTS).SMARTS

The semantics of the "zero-level" parentheses are that all of the atom and bond expressions within a set of zero-level parentheses must match within a single component of the target.

SMARTS	SMILES	Match behavior
C.C	CCCC	yes, no component level grouping specified
(C.C)	CCCC	yes, both carbons in the query match the same component
(C).(C)	CCCC	no, the query must match carbons in two different components
(C).(C)	CCCC.CCCC	yes, the query does match carbons in two different components
(C).C	CCCC	yes, both carbons in the query match the same component
(C).(C).C	CCCC.CCCC	yes, the first two carbons match different components, the third matches a carbon anywhere

These component-level grouping operators were added specifically for reaction processing. Without this construct, it is impossible to distinguish inter- versus intramolecular reaction queries. For example:

Reaction SMARTS expression	Match behavior
C(=O)O.OCC>>C(=O)OCC.O	Matches esterifications
(C(=O)O).(OCC)>>C(=O)OCC.O	Matches intermolecular esterifications
(C(=O)O.OCC)>>C(=O)OCC.O	Matches intramolecular esterifications (lactonizations)

4.6 Reaction Queries

Reaction queries are expressed using the SMARTS language. SMARTS has been extended to handle reaction query features in much the same fashion as SMILES has been extended to handle reactions.

A reaction query may be composed of optional reactant, agent, and product parts, which are separated by the ">" character. In this case, the parts of the reaction query match against the corresponding roles within the reaction target, as expected. Note that it is also quite reasonable to search a set of reactions by giving a molecule query. In this case, the answer is a hit if the molecule SMARTS matches anywhere within the reaction target. In effect, matching a molecule SMARTS against a reaction target is a query where the role of the SMARTS is unspecified.

Example Reaction SMARTS:		
Query:	Target:	Matches:
C>>	CC>>CN	2

>C>	CC>>CN	0
>>C	CC>>CN	1
C	CC>>CN	3

The atom mapping for a reaction query is optional. When included in the definition of the pattern, it is used for searching.

If atom maps are used for a SMARTS match, their only effect is to potentially eliminate answers from the result. Atom maps can never, under any circumstance cause the addition of hits to an answer set. Conceptually, one can consider the atom map matching as a post-processing step after a "normal" match. Each of the hits is examined to make sure the atom map classes match on the reactant and product sides of the reaction.

In SMARTS, the atom map has unusual semantics. An atom map is a property which must be evaluated on a global scope during the match. One can not know if the map is correct without considering every atom in the match, in effect requiring the enumeration of every possible path before testing. This is much more computationally expensive than the current SMARTS implementation, which tests the paths as they are built and stops as soon as a path fails to match.

In order to avoid this computational trap, the expressiveness of SMARTS for atom maps has been limited to a low-precedence and operation. That is, only expressions of form: "[expr:n]" or "[expr:?n]" are allowed, where "expr" is any legal atomic expression excluding atom maps and "n" is a map class value. This expression is a low-precedence logical AND between "expr" and the map expression ":n". The following examples illustrate other nuances of the semantics:

Example Reaction SMARTS:			
Query:	Target:	Matches:	Comment:
C>>C	CC>>CC	4	No maps, normal match.
C>>C	[CH3:7][CH3:8]>> [CH3:7][CH3:8]	4	No maps in query, maps in target are ignored.
[C:1]>>C	[CH3:7][CH3:8]>> [CH3:7][CH3:8]	4	Unpaired map in query ignored.
[C:1]>>[C:1]	CC>>CC	0	No maps in target, hence no matches.
[C:?1]>>[C:?1]	CC>>CC	4	Query says mapped as shown or not present.
[C:1]>>[C:1]	[CH3:7][CH3:8]>>[CH3:7][CH3:8]	2	Matches for target 7,7 and 8,8 atom pairs.
[C:1]>>[C:2]	[CH3:7][CH3:8]>> [CH3:7][CH3:8]	4	When a query class is not found on both sides of the query, it is ignored; this query does NOT say that the atoms are in different classes.
[C:1][C:1]>>[C:1]	[CH3:7][CH3:7]>> [CH3:7][CH3:7]	4	Atom maps match with "or" logic. All atoms get bound to class 7.
[C:1][C:1]>>[C:1]	[CH3:7][CH3:8]>> [CH3:7][CH3:8]	4	The reactant atoms are bound to classes 7 and 8. Note that having the first query atom bound to class 7 does not preclude binding the second atom. Next, the product

			atom can bind to classes 7 or 8.
[C:1][C:1]>>[C:1]	[CH3:7][CH3:7]>> [CH3:7][CH3:8]	2	The reactants are bound to class 7. The product atom can bind to class 7 only.

The last example is the most confusing. Since there is no "or&quot logic for atom maps, the behavior when checking the maps is as follows: the query reactants can be bound to any classes in the target. These bindings form the set of allowed product bindings. The product query atoms are then tested against this list. If all of the product atoms pass, then the path is a match. The effect of this procedure is to provide the "logical-OR" semantics for atom maps within the simple implementation. The downside of this implementation is that it can be confusing to the user. Fortunately, the simple pairwise atom maps will suffice for most users.

Finally, atom map labels in molecule SMARTS and unpaired atom map labels in reaction SMARTS are ignored. Stated another way, since the atom maps express the idea of a global association of atoms across a reaction, atom maps on a molecule query have no meaning. Similarly, a lone atom map on a reaction atom which doesn't correspond to any other atoms in the query has no meaning. In both of these cases, the query is identical to the query written without the meaningless atom maps.

In recursive SMARTS, reaction expressions are not allowed. The reasons for this are twofold: first, it isn't clear that the meaning of a recursive SMARTS for a reaction would have any useful expressiveness and second, there is a practical problem with the lexical definitions of reactions: given the strict left-to-right definition of reactant-agent-product, how would one express a product atom in a vector binding?? Of course we can change the syntax for recursive SMARTS or reactions to accommodate this if it becomes clear that it is useful.

4.7 SMARTS Versus SMILES

All SMILES expressions are also valid SMARTS expressions, but the semantics changes because SMILES describes molecules whereas SMARTS describes patterns. The molecule represented by a SMILES string is usually, but not always, matched by the same string when used as a SMARTS.

SMILES is interpreted as a molecule, and it is the resultant molecule (not the SMILES string) which is subject to searching. Similarly, SMARTS is interpreted as a pattern; it is this pattern (not the SMARTS string) which is matched against molecules. For instance, the SMILES "C1=CC=CC=C1" (cyclohexatriene) is interpreted as the benzene molecule. This molecule will be matched by the SMARTS c1ccccc1, which is interpreted as the pattern "6 aromatic carbons in a ring". The SMARTS "C1=CC=CC=C1" makes a pattern ("six aliphatic carbons in a ring with alternating single and double bonds") which will *not* match benzene. It will, however, match the nonaromatic phenylate cation with SMILES C1=CC=CC=[CH+]1.

When atoms are specified without brackets in SMILES, default values are used; in SMARTS, unspecified properties are not defined to be part of the pattern. For instance, the SMILES O means an aliphatic oxygen with zero charge and two hydrogens, i.e. water. In SMARTS, the same expression means any aliphatic oxygen regardless of charge, hydrogen count, etc, e.g. it will match the oxygen in water, but also those in ethanol, acetone, molecular oxygen, hydroxy and hydronium ions, etc. Specifying [OH2] limits the pattern to match only water (this is also the fully specified SMILES for water).

There are a few anachronisms in most SMILES interpreters which can also lead to confusion. Some SMILES interpreters allow implicit hydrogens to be added as explicit atoms on input as a shortcut. E.g., the SMILES for 1H-pyrrole is "[nH]1cccc1" which is matched by itself as SMARTS and by "n1cccc1". The current Daylight SMILES interpreter will also accept "Hn1cccc1" for (not very good) reasons of historical compatibility; this generates the same (hydrogen-suppressed) molecule as does "[nH]1cccc1" and is matched

by the same SMARTS. However, the SMARTS "Hn1cccc1" does not match this molecule.

Most SMARTS expressions are not valid SMILES expressions. For instance, the string "cOc" is a valid SMARTS, matching an aliphatic oxygen connected to two aromatic carbons as part of a larger molecule (e.g. diphenyl ether). However, "cOc" does not describe a molecule per se, and is therefore not a valid SMILES.

4.8 Efficiency Considerations

The Daylight 4.x SMARTS Toolkit provides a function, `dt_smarts_opt()`, which automatically optimizes a SMARTS by reordering, expanding, and/or consolidating atom and bond expressions. Programs which use this feature (e.g. the Merlin program) can be expected to be near optimal in terms of the time used to search typical organic structures.

When this optimization method is not used, there are some things which can be done to facilitate efficient (fast) searching operations using SMARTS. It is important to recognize that SMARTS target strings are processed in strictly left-to-right order. For this reason, substantial gains in speed can be achieved by following these guidelines:

- Uncommon atoms or bond arrangements should be placed early in SMARTS targets.
- In an "and-expression", the less common atom or bond specifications should be placed early.
- In an "or-expression", the less common atom or bond specifications should be placed last.

4.9 Examples

cc	any pair of attached aromatic carbons
c:c	aromatic carbons joined by an aromatic bond
c-c	aromatic carbons joined by a single bond (e.g. biphenyl).
O	any aliphatic oxygen
[O;H1]	simple hydroxy oxygen
[O;D1]	1-connected (hydroxy or hydroxide) oxygen
[O;D2]	2-connected (etheric) oxygen
[C,c]	any carbon
F,Cl,Br,I]	the 1st four halogens.
[N;R]	must be aliphatic nitrogen AND in a ring
[!C;R]	(NOTaliphatic carbon) AND in a ring
[n;H1]	H-pyrrole nitrogen
[n&H1]	same as above
[c,n&H1]	any arom carbon OR H-pyrrole nitrogen
[c,n;H1]	(arom carbon OR arom nitrogen) and exactly one H
!@	two atoms connected by a non-ringbond
@;!:	two atoms connected by a non-aromatic ringbond
[C,c]=,#[C,c]	two carbons connected by a double or triple bond

5. SMIRKS - A Reaction Transform Language

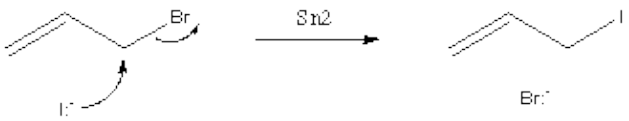
A transform is simply a generic reaction within the Daylight system. Generic reactions are extremely useful for chemical information processing. They can be used to create new reactions, manipulate molecules, and to generate new molecules on a large (conbinatorial) scale. They are somewhat complicated, because they must meet several conflicting sets of requirements. These requirements, and how the Daylight system addresses them, are discussed here.

5.1 Description

A complete reaction can be described in a number of ways: the reactant/product notation used in SMILES, as a reaction graph, and as a atom- and bond-change list plus the reactant molecules. It is straightforward to interconvert any of the three representations of a complete reaction.

The most intriguing way to think about a complete reaction is as an atom and bond change list. This representation can be easily understood to capture the idea of a generic reaction. Any set of reactions which undergo the same set of atom and bond changes, regardless of the underlying molecule substrates, can be considered an example of a given generic reaction.

Consider once again our Sn2 reaction. If it is a normal Sn2 reaction, then the list of atom and bond changes during the reaction is as follows:



Reaction Change List:		
Part:	Change Type:	Change:
C-Br	Bond	Single bond -> no bond
C-I	Bond	no bond -> Single bond
Br	Atom	no charge -> -1 charge
I	Atom	-1 charge -> no charge

Any reaction which undergoes the same set of atom and bond changes would be considered part of the same generic reaction. For example, reaction of Potassium Iodide rather than Sodium Iodide, or reaction of any alkyl bromide in place of Allyl Bromide all have the same list of atom and bond changes. Note that this bare-bones representation of the reaction does not take into account other factors which might affect the reaction such as the steric effects of a primary bromide versus a secondary or tertiary one, and the electronic activating effect of the allylic bond.

As an aside, note the similarities of this representation with the "Difference Fingerprints" described previously. In effect, the difference fingerprint is calculated directly from the bond changes during a reaction (atom property changes like charge, stereochemistry, are not included). The difference fingerprints will be identical for all examples of a single generic reaction.

So, there are two distinct requirements to accurately capture a generic reaction. First is the actual set of changes to the molecule which occur during the reaction (captured with the atom and bond changes) and

second is the indirect effects of activating and deactivating groups near the reaction site.

Within the Daylight system, the indirect effects on a generic reaction are most appropriately expressed with the SMARTS query language. With it, one can express concepts such as "electron-withdrawing group", "electron-donating group", aromaticity, unsaturation, steric effects, etc.

The parallels here should be evident: a complete reaction consists of a set of atom and bond changes, plus the substrate molecule upon which the changes operate. A generic reaction consists of the *same* set of atom and bond changes, plus a substrate SMARTS pattern upon which the changes operate. Any molecule which matches the SMARTS pattern is a candidate for the generic reaction.

5.2 Representation

In the Daylight System, we adopted the reactant/product notation for generic reactions. It is not as compact as a reaction graph, but it is the most compatible and most consistent with the SMILES and SMARTS languages already defined for reactions.

The language SMIRKS is defined for generic reactions. It is a hybrid of SMILES and SMARTS in order to meet the dual needs for a generic reaction: expression of a reaction graph and expression of indirect effects. It is a restricted version of reaction SMARTS involving changes in atom-bond patterns. The rules for SMIRKS are:

- The reactant and product sides of the transformation are required to have the same numbers and types of mapped atoms and the atom maps must be pairwise. However, non-mapped atoms may be added or deleted during a transformation.[1]
- Stoichiometry is defined to be 1-1 for all atoms in the reactant and product for a transformation. Hence, if non-unit stoichiometry is desired, reactants or products must be repeated.
- Explicit hydrogens that are used on one side of a transformation must appear explicitly on the other side of the transformation and must be mapped.
- Bond expressions must be valid SMILES (no bond queries allowed).
- Atomic expressions may be any valid atomic SMARTS expression for nodes where the bonding (connectivity & bond order) doesn't change.[1] Otherwise, the atomic expressions must be valid SMILES.

[1] Introduced in v4.6.1.

These above rules guarantee that the SMIRKS can be interpreted as a reaction graph and that the atom and bond changes can be derived from this representation. This set of rules satisfies the first requirement for a generic reaction. The final rule allows the expression of the "indirect effects" of a generic reaction for atoms which don't participate directly in the reaction.

The net result of these rules is a language which can capture the ideas of generic reactions. A number of examples follow to illustrate features of the language.

5.3 Transform Grammar

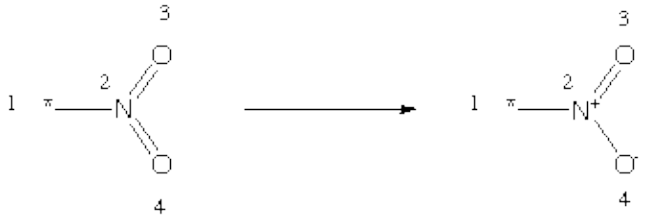
```
transform               :       reactant '>' agent '>' product
                        |       reactant '>>' product
                        ;
reactant,
agent,
```

```
product                 :       pattern
                        ;
pattern                 :       SSMARTS
                        ;

SSMARTS:  a valid pattern specification, excluding bond expressions, and
          using a limited set of atom expressions (Subset-SMARTS).
```

5.4 Examples

First is a simple transform to interconvert nitro-group representations in the toolkit. The nitro group is typically represented either with pentavalent Nitrogen "*N(=O)=O" or as the charge-separated trivalent Nitrogen "*[N+](=O)[O-]". These can be interconverted with the following transform:



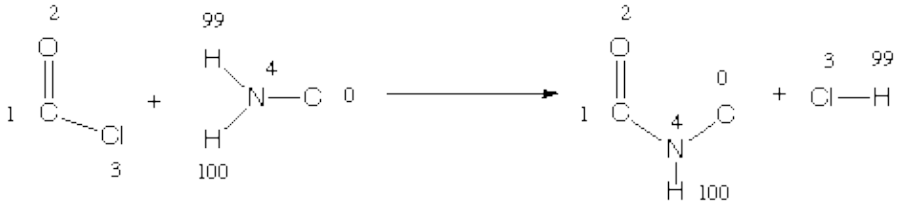
[*:1][N:2](=[O:3])=[O:4]>>[*:1][N+:2](=[O:3])[O-:4]

This transform illustrates an important point: transforms need not represent real reactions. Transforms are useful as a general tool for manipulation of molecules in the toolkit. Most atom and bond changes can be written as legal transformations. Hence, transforms become a powerful tool in the chemist/programmers arsenal for chemical information processing. Also, as with any transform, this one can be used in either the forward or reverse direction.

Inspection of the transform indicates that this meets the requirements for a legal transform. First, it has the same number of atom expressions on both sides of the transform, and they are mapped pairwise. The atom expressions are all legal SMILES and the bond expressions are all legal SMILES.

In this example, there are no SMARTS expressions found. SMARTS atomic expressions could be substituted for the atoms of map classes ":1" and ":3" only. The two atoms attached to the bond which changes ("N:2" and "O:3") must be expressed as SMILES. The change in valence and charge which occurs can be deduced unambiguously from the SMIRKS. Were atomic expressions allowed for these nodes, the determination of atomic properties might not be possible.

The next example illustrates the most confusing part of the SMIRKS language, which is the handling of hydrogens. Unfortunately, the SMILES and SMARTS languages express hydrogens inconsistently. These inconsistencies have been partially reconciled in the SMIRKS language by first, requiring that all hydrogens directly involved in a transform (bonds change) must be expressed explicitly and second, changing the meaning of SMARTS for a single case: [H]. There are still some cases which will cause confusion, however.



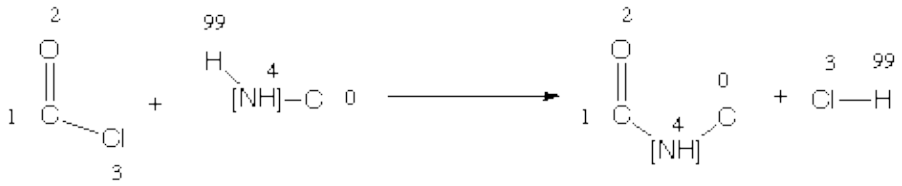
[C:1](=[O:2])[Cl:3].[H:99][N:4]([H:100])[C:0]>> [C:1](=[O:2])[N:4]([H:100])[C:0].[Cl:3][H:99]

Note that both hydrogens attached to the nitrogen of the reaction are shown as explicit. Based on the SMIRKS rules, the expression [H:99] must be interpreted as SMILES, since the bonding to this node changes during the reaction. The expression [H:100] may be interpreted as SMARTS, since its bonding does not change in the reaction. Recall that in versions of the Daylight system prior to 4.51, [H] as SMARTS meant: "any atom with a single attached hydrogen", while in SMILES it is a lone explicit hydrogen.

These differences in interpretation would make SMIRKS unintelligible. Hence, a single change to SMARTS interpretation, for expressions of the form: [<weight>]H<charge><map>]. In SMARTS, these expressions now are interpreted as a hydrogen atom, rather than as any atom with one hydrogen attached. All other SMARTS hydrogen expressions retain their pre-4.51 meanings.

SMARTS/SMIRKS hydrogen expressions:		
Expression:	4.42 meaning:	4.51 meaning:
[H]	Atom with one attached hydrogen	A hydrogen atom
[#1]	A hydrogen atom	Unchanged
[H1]	Atom with one attached hydrogen	Unchanged
[*H]	Atom with one attached hydrogen	Unchanged
[H,+] or [*,H], etc.		Unchanged

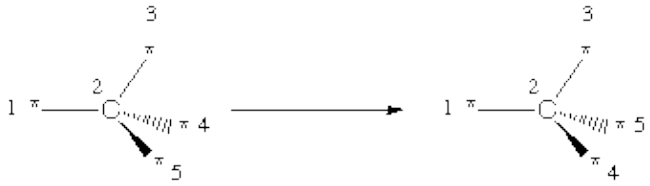
The result of the change in semantics is that both explicit hydrogens in the example SMIRKS are interpreted consistently as hydrogen atoms. Note that there still may be confusion for 'implicit' hydrogens. For example, if the amide formation reaction were expressed as:



[C:1](=[O:2])[Cl:3].[H:99][NH:4][C:0]>> [C:1](=[O:2])[NH:4][C:0].[Cl:3][H:99]

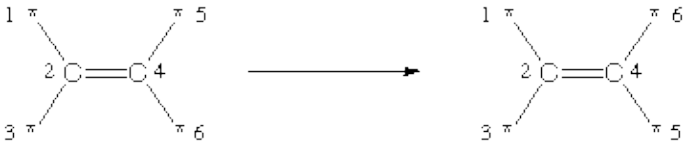
This case only matches secondary amines. The expression [NH:4] matches a nitrogen with exactly one hydrogen attached (the [H:99] is it). Hence, any other attachments must be non-hydrogen. In general for SMIRKS, the best strategy for expressing hydrogens is to include them as explicit atoms if they are involved in the reaction directly or if they are attached to atoms which are involved in the reaction. This will eliminate most of the confusing cases.

Stereochemistry in SMIRKS is handled locally based on atom map labels. That is, a stereochemical specification describes the orientation of atoms or bonds based solely on the ordering in the string and the atom map labels. For example:



[*:1][C@:2]([*:3])([*:4])[*:5]>>[*:1][C@@:2]([*:3])([*:4])[*:5]
[*:1][C@:2]([*:3])([*:4])[*:5]>>[*:1][C@:2]([*:4])([*:3])[*:5]

This inverts any carbon stereocenter encountered. On the reactant side of the transform, the expression describes a specific orientation of the nodes; similarly, on the product side the inverted orientation of the same nodes is described. Similarly, for bond stereochemistry:



[*:1]/[C:2]([*:3])=[C:4]([*:5])/[*:6]>>[*:1]/[C:2]([*:3])=[C:4]([*:5])\[*:6]

This inverts any C=C double-bond stereochemistry matched. Note that both of the above examples can match a single stereocenter multiple ways, however the net result is always an inversion of the stereocenter based on the specification of the transform.

In general, transforms which involve stereochemistry should be written with sufficient context for the toolkit to interpret the local chirality needed for analysis. For tetrahedral chirality, all four connections to the chiral atom should be explicitly shown and for double-bond chirality, all three connections to each atom (one double-bond and two single bonds) should be shown.

Finally, a point about the new component-level grouping operators in SMARTS and SMIRKS. This syntax allows the expression of inter- and intramolecular reactions in both SMARTS and SMIRKS. This syntax is fully supported in SMIRKS. See the section on SMARTS section on Reaction Queries for more information.

6. Fingerprints - Screening and Similarity

Similarity measures, calculations that quantify the similarity of two molecules, and *screening*, a way of rapidly eliminating molecules as candidates in a substructure search, are both processes that use *fingerprints*. Fingerprints are a very abstract representation of certain structural features of a molecule; before we describe them, we'll discuss the problems that inspired the development of the fingerprinting techniques used in the Daylight Chemical Information System. To begin, let us define a few terms:

pattern
: The thing for which you are searching; also called a *target* or *substructure*.

molecule
: The thing being searched. Sometimes called the *object* or *structure*.

N

The "size" of a pattern. Roughly speaking, equal to the number of atoms and bonds the pattern contains (its exact definition isn't important, only the general idea).

P «in» M

We use the «in» symbol to mean "is a substructure of" - i.e. "P «in» M" means "pattern P is a substructure of molecule M". "P «not in» M" means "pattern P is not a substructure of molecule M".

O(f(N))

The *order* (computation time, or "cost") of performing a search. For example, $O(N^2)$ (pronounced "order N squared") means that the running time is proportional to the square of a pattern's size.

Substructure searching is known to be in the *non-polynomial-complete (NP-complete)* class of computational problems. *Non polynomial* means that the worst-case time to solve such a problem can never be expressed as a polynomial in which the number of atoms and/or bonds is the independent variable; i.e. it can't be of the form $O(N^K)$. Instead, the worst-case time for a substructure search is always of the form $O(K^{kN})$.

This is quite unfortunate. To see why, imagine (for simplicity) that a substructure-search program takes $2^N/10^6$ seconds to compute where N is the number of atoms. This program can solve a 1-atom problem in a microsecond, and a 10-atom problem takes about a millisecond. This does not seem too bad until we realize that each time we add an atom we double our time: a 20-atom problem takes about a second, and a 30-atom problem takes 17 minutes. Clearly this algorithm will be inadequate as we attempt to solve real chemical problems. By contrast, if we could find an algorithm that ran in $N^2/10^3$ seconds, it would be 1000 times slower on the 1-atom problem but could solve the 30-atom problem in less than a second. Clearly a polynomial solution is better than an exponential solution.

Luckily, although substructure searching is exponential in the worst case, real chemicals don't exhibit the high connectivity which, in a generalized mathematical graph, leads to worst-case behavior. Typical chemical substructure searches take $O(N^2)$ or $O(N^3)$ time, and although this is not exactly blazingly fast, it is a lot better than exponential behavior. But even this polynomial performance is slow - it can take a significant fraction of a second for a substructure search - and NP-complete theory tells us that we might occasionally run into the worst-case, an exponential-time search.

One of the cold, hard facts about NP-complete problems is that there is no way around them. If you think you've found a substructure-search algorithm that *always* runs in polynomial time, you should try your hand at a perpetual-motion machine. Your algorithm might work for most cases, but if it always finishes in polynomial time some of its answers must be wrong.

Fortunately there is a "hole" of sorts in these cold, hard facts: we can't detect the *presence* of a substructure in polynomial time, but we can often detect the *absence* of a substructure much faster, often *linear* time: O(N). The trick is to use an "imperfect" algorithm, one that can say **P «not in» M** with 100% confidence but that can only say **P «in» M** with lower confidence Such an algorithm, called a *screen*, does not violate any mathematical laws - ultimately we still have to use a real substructure search to get a 100%-confident **P «in» M** answer - but our "cheap" algorithm screens out most cases, avoiding the "expensive" algorithm most of the time.

6.1 A Brief History of Screening Large Databases

Many substructure-searching problems call for repeatedly examining a large number of molecules (typically stored in a database), comparing each with a pattern. In such situations, it pays to spend some time "up front," storing the answers to specific questions for each structure in the database. Subsequent searches of the database use these pre-computed answers to vastly improve search time; the up-front computation time is paid

back quickly as repeated searches are performed.

For example, one simple screen notes the molecular formula (MF) of each molecule as it is added to the database. When a pattern is presented for searching, we generate its molecular formula; during the search, we compare the pattern's MF to each molecule's MF, and reject any molecules that are missing atoms the pattern requires. By doing this, we eliminate expensive substructure searches that are doomed to fail for the "obvious" reason of not having enough of a particular element. If the MF screen says **P «not in» M**, we can be 100% confident that it is correct; if the molecular formulas are compatible we have to continue with other screens or with the substructure search itself.

Molecular Formula is only one of many screens we can apply, but it illustrates the fundamental concept of screening: We only do the "expensive" substructure search when no screen can say **P «in» M**. By devising clever screens, we can increase the reliability of the screens to where they reject almost all structures except those that ultimately pass the substructure test (that is, the screens have very few "false positives").

6.1.1 Structural keys

Structural keys were the first type of screen employed for high-speed screening of chemical databases. A structural key is usually represented as a *boolean array*, an array in which each element is TRUE or FALSE. Boolean arrays in turn are usually represented as *bitmaps*, an array of bytes or words in which each bit represents one position of the boolean array. As the name implies, a *structural key* is a bitmap in which each bit represents the presence (TRUE) or absence (FALSE) of a specific structural feature (pattern).

To make a structural key, one decides which structural features (patterns) are important, assigns a bit of the bitmap to each, then generates a bitmap for each molecule in the database. Generating a structural key is time-consuming: you have to do a substructure search for *each* pattern represented in the bitmap, and repeat this for *each* molecule in the database.

The list of patterns that one might use is long. Some examples are:

- The presence/absence of each element, or if an element is common (nitrogen, for example), several bits might represent "at least 1 N", "at least 2 N", "at least 4 N", and so forth.
- Unusual or important electronic configurations, such as "sp^3 carbon" or "triple-bonded nitrogen."
- Rings and ring systems, such as cyclohexane, pyridine, or napthalene.
- Common functional groups, such as alcohols, amines, hydrocarbons, and so forth.
- Functional groups of special importance in a particular database. For example, a database of organo-metallic molecules might have bits assigned for metal-containing functional groups; in a drug database one might have bits for specific skeletal features such as steroids and barbiturates.
- "Disjunctions" of unusual features. There might be patterns that are particularly rare, thus not individually worth the "cost" of a bit, yet extremely significant when they do occur. Several such patterns can be assigned to the same bit; if any one of the patterns is present the bit is set.

When a database is to be searched for a particular pattern, a structural key is generated for the pattern. As the search proceeds, the pattern's structural key is compared to that of each molecule in the database. If any TRUE bit in the pattern's key is not also TRUE in the molecule's key, then the feature represented by that bit is not in the molecule, and the pattern couldn't possibly be a substructure of the molecule. Structural keys

make a very fast screen for substructure searching since computers perform the necessary boolean operations very quickly.

Structural keys vary widely in size, from a few tens or hundreds of bits to several thousand bits (in a single database, structural keys are usually all the same size since they all must represent the same thing). The choice of size is a tradeoff between specificity and space: The more bits there are, the better the chances that the screen can say **P «in» M** and thus avoid a full substructure search.

6.1.2 Fingerprints

The next evolutionary step in high-speed structural screening was the *fingerprint*, a more abstract relative of the structural key.

The structural keys described above suffer from a lack of generality. The choice of patterns included in the key has a critical effect on the search speed across the database: An effective choice will screen out virtually all structures that aren't of interest, greatly increasing search speed, whereas a poor choice will cause many "false hits," which slows searching to a crawl. The choice of patterns also depends on the nature of the queries to be made: A structural key used by a group of pharmaceutical researchers might be nearly worthless to a group of petrochemical researchers.

Fingerprints address this lack of generality by eliminating the idea of pre-defined patterns. A fingerprint is a boolean array, or bitmap, but unlike a structural key there is no assigned meaning to each bit. Your own fingerprint is very characteristic of you, yet there is no meaning to any particular feature. Similarly, a pattern's fingerprint characterizes the pattern, but the meaning of any particular bit is not well defined.

Unlike a structural key with its pre-defined patterns, the patterns for a molecule's fingerprint are generated from the molecule itself. The fingerprinting algorithm examines the molecule and generates the following:

- a pattern for each atom
- a pattern representing each atom and its nearest neighbors (plus the bonds that join them)
- a pattern representing each group of atoms and bonds connected by paths up to 2 bonds long
- ... atoms and bonds connected by paths up to 3 bonds long
- ... continuing, with paths up to 4, 5, 6, and 7 bonds long.

For example, the molecule **OC=CN** would generate the following patterns:

0-bond paths:	**C**	**O**	**N**
1-bond paths:	**OC**	**C=C**	**CN**
2-bond paths:	**OC=C**	**C=CN**	
3-bond paths:	**OC=CN**		

The list of patterns produced is exhaustive: *Every* pattern in the molecule, up to the pathlength limit, is generated. For all practical purposes, the number of patterns one might encounter by this exhaustive search is infinite, but the number produced for any *particular* molecule can be easily handled by a computer.

Because there is no pre-defined set of patterns, and because the number of possible patterns is so huge, it is not possible to assign a particular bit to each pattern as we did with structural keys. Instead, each pattern serves as a seed to a pseudo-random number generator (it is "hashed"), the output of which is a set of bits (typically 4 or 5 bits per pattern); the set of bits thus produced is added (with a logical `OR`) to the fingerprint.

Note that because each set of bits is produced by a pseudo-random generator, it is likely that sets will overlap. For example, suppose we are in the middle of generating a fingerprint, and it happens that 1/4 of the bits are already set. If the next pattern generates a set containing 5 bits, the probability that all 5 bits will be unique is $(3/4)^5$, or about 24%. Likewise, the probability that all 5 bits will *not* be unique are $(1/4)^5$, or about 0.1%.

In spite of the difference between the meaning of a fingerprint's bits and a structural key's bits, fingerprints share an important feature with structural keys: If a pattern is a substructure of a molecule, *every bit that is set in the pattern's fingerprint will be set in the molecule's fingerprint.* This means that, like structural keys, we can use simple boolean operations on fingerprints to screen molecules as we search a database, making a fingerprint comparison an extremely fast screen for substructure searching.

The best way to think of the bits of a fingerprint is as "shared" among an unknown but very large number of patterns. Each pattern generates its particular set of bits; so long as at least one of those bits is unique (not shared with any other pattern present in the molecule), we can tell if the pattern is present or not. A structural key indicates with certainty that a particular pattern is present or absent. Fingerprints are not so definite: if a fingerprint indicates a pattern is missing then it certainly is, but it can only indicate a pattern's presence with some probability. Although a fingerprint doesn't indicate with 100% certainty that a particular pattern is present, it contains far more patterns total than a structural key, the net result being that a fingerprint is a far better screen than a structural key in almost all situations.

Fingerprints have several advantages over structural keys:

- Since fingerprints have no pre-defined set of patterns, one fingerprinting system serves all databases and all types of queries.
- More effective use is made of the bitmap. Structural keys are usually very "sparse" (mostly zeros) since a typical molecule has very few of the patterns that the structural key's bits represent. Although a mathematical analysis of fingerprint density is beyond the scope of this introduction, it turns out that fingerprints can be relatively "dense" (20-40% ones) without losing specificity. The result is that a fingerprint can be much smaller than a structural key with the same discriminating power.
- The patterns that go into a fingerprint are highly overlapped - except for "lone atoms", each pattern shares portions of itself with at least one other pattern (the example above illustrates this). The result is that the more complex a molecule gets, the more accurately its fingerprint characterizes it.

6.1.3 Variable-sized Fingerprints

The next evolutionary step in screening was the concept of *folding* a fingerprint to increase information density.

In the discussion above we mentioned the *sparseness* of a fingerprint, which is directly related to its *information density*. A fingerprint's information density can be thought of as the ratio how much information it actually holds to how much it could hold. As a practical definition, we measure the *bit density*, the ratio of "on" bits to the total number of bits (e.g. the bit density of "11000000" is 0.25).

Fingerprints for small molecules and "featureless" molecules (such as **CH4** or **C40H82**) have less information in them than those for large or "rich" molecules. But the fingerprinting mechanisms discussed so far require a fixed fingerprint size for all molecules. If we choose to use small fingerprints, the fingerprint of large or complex molecules will be "black" - nearly all ones - and will not discriminate well (there is more information than the fingerprint can hold). On the other hand, if we use very large fingerprints, most molecules' fingerprints will be "white" - nearly all zeros - and will waste space. In both cases we have low information density; the "black" fingerprint because it is too dense and the "white" one because it is too sparse.

Ideally, we would like to choose a particular discriminatory power (e.g. "For a typical pattern, 98% of the database is screened out by the pattern's fingerprint") and compute the fingerprint density needed to achieve that discriminatory power on a case-by-case basis. Although we can not actually do this, the process of "folding" achieves nearly the same performance and size as the "ideal" case.

The folding process begins with a fixed fingerprint size that is quite large - large enough to accurately represent any molecule we expect to encounter. The fingerprint is then *folded*: we divide it into two equal halves then combine the two halves using a logical OR. The result is a shorter fingerprint with a higher bit density. We can repeatedly fold the fingerprint until the desired information density (called the *minimum density*) is reached or exceeded.

As long as two fingerprints are the same size (even if created with different sizes), they are compatible. To see why, consider the fingerprints of a pattern P and a molecule M. If the screen is initially positive (e.g. all bits in the P's fingerprints are also in M's) then the same will be true after folding. On the other hand, a negative screen (at least one bit in P's fingerprint is not in M's) might be converted to a positive screen after folding. But this is ok - converting "correct negative" to a "false positive" doesn't violate the rules of screening: a screen is only required to say **P «in» M** with 100% reliability. With each fold, we increase the chances of a false positive but save half of the space needed to store the fingerprint.

Fingerprint folding allows us to optimize the information density in a set of fingerprints, thus optimizing screening speed. Rather than choosing one fingerprint size for the entire database, we choose the size of each molecule's fingerprint individually, according to the complexity of the molecule and the desired success rate of the screening process. In most real databases, optimizing the information density greatly reduces the amount of data stored, and increases the screening speed correspondingly.

6.1.4 In-memory Screening

Although there is no reason a structural key can't reside in memory rather than on disk, they tend to be considerably larger than fingerprints, too large to fit a reasonable number of them into memory. This was especially true when structural keys were first developed - memory was several orders of magnitude more expensive than it is now, and most computers had precious little of it. Thus, structural-key screening has almost always been done as a disk-based screening technique.

Computer memories are roughly 10^5 times faster than disks. If one can read the bitmaps of a structural key or fingerprint into memory and search it there, the speed of the screen itself increases a corresponding amount. About the time fingerprints were developed, computer memory prices reached a point where the fingerprints from a relatively large database all could be loaded into memory.

With the advent of in-memory fingerprint-based screening, exploratory data analysis takes on a completely new aspect. One can quite literally "explore" a database, trying different searches, refining them, taking side-tracks to see where they would lead, and so on.

The evolution of the above-described screening techniques has now reached the point where the SMILES and fingerprints of all chemicals known in the world (roughly 10-15 million structures) can fit in the memory of a large workstation-class computer; it doesn't even take a "mainframe" computer, much less a supercomputer. An ordinary database of tens or hundreds of thousands of molecules can easily fit into the memory of today's run-of-the-mill workstations. Speed increases are correspondingly dramatic: an ordinary workstation can screen 100,000 to 1,000,000 structures per second using in-memory folded fingerprints.

The number of molecules that can be considered in a single search has also grown dramatically. It is now actually possible to search all known chemicals in the world in a reasonable time (that is, if you can get your hands on such a database). Most users, using a corporate or academic database of less than a million molecules, will be able to search their database in a few seconds - the time it takes to answer the question is now much shorter than the time it takes to think of the question, even on relatively large databases.

6.2 Fingerprints and Reactions

There are two distinct types of fingerprints which the Daylight system provides for reaction processing. First is the "normal" structural fingerprint, which is useful as a superstructure search screen. Second is the "difference" fingerprint, which is useful as a metric for the bond changes which occur during a reaction.

6.2.1 Structural Reaction Fingerprints

The structural reaction fingerprint is nothing more than the combination of the normal structural fingerprints for the reactant and product molecules within the reaction. The combination is the bitwise-OR of the following fingerprints:

- Fingerprint of the reactant part
- Fingerprint of the product part
- Bit-shifted fingerprint of the product part

This behavior allows the fingerprint to serve as a structural screen for superstructure-matching and allows the fingerprint to provide some discrimination power between reactant and product parts. Note that the agent molecules are not used in the structural fingerprint.

Once the structural reaction fingerprint has been generated, all of the normal fingerprint operations (folding, similarity, substructure screen) apply.

6.2.2 Reaction Difference Fingerprints

The difference fingerprint is a new type of fingerprint specifically tailored for reaction processing. If one has a reaction which is stoichiometric (all atoms on the reactant side appear on the product side), then the difference in the fingerprint of the reactant molecules and the fingerprint of the product molecules will reflect the bond changes which occur during the reaction.

On the surface, one could perform an "exclusive-OR" operation between the reactant and product fingerprint. Unfortunately, since the molecule fingerprints do not keep track of the count of paths many of the relevant bonds get masked out during the "XOR" operation. What is required is to keep track of the count of each path in the reactant and product and then subtract the counts of a given path. If the difference in count is non-zero, then the path is used to set a bit in the difference fingerprint. If the difference in count is zero, then no bit is set for that path in the difference fingerprint.

For example, consider our well-worn Sn2 displacement reaction:

[I-].[Na+].C=CCBr>>[Na+].[Br-].C=CCI

The paths generated for the molecules would be as follows:

Enumerated Fingerprint Paths:		
Path Length:	Reactant (count/path):	Product (count/path):
0	1 I, 1 Na, 3 C, 1 Br	1 I, 1 Na, 3 C, 1 Br
1	1 C=C, 1 C-C, 1 C-Br	1 C=C, 1 C-C, 1 C-I
2	1 C=C-C, 1 C-C-Br	1 C=C-C, 1 C-C-I
3	1 C=C-C-Br	1 C=C-C-I

Difference in Path Counts:	
Path Length:	Difference (count/path):
0	0 I, 0 Na, 0 C, 0 Br
1	0 C=C, 0 C-C, 1 C-Br, 1 C-I
2	0 C=C-C, 1 C-C-Br, 1 C-C-I
3	1 C=C-C-Br, 1 C=C-C-I

After generating the difference in counts, we only use the six paths with non-zero differences to set bits in the difference fingerprint. These are the paths which walk through bonds that change during the reaction. By considering only these paths, we get a fingerprint which reflects the overall bond changes in the reaction.

Once generated, the difference fingerprint is a normal fingerprint and can be folded, used for similarity measurements and clustering. The difference fingerprint may not be used as a substructure screen for any types of searching, since it does not obey the strict subset relationship required for screening. In the above example, note that the "[Na+]" atom does not appear in any of the difference fingerprint paths. Thus, any search query which contained Sodium-containing paths would not pass the screening step, even if a valid query match.

6.3 Similarity Measures

One often wishes to know how similar one molecule is to another, independent of whether either is a substructure of the other. There are many ways one might choose to measure such similarity; for example a chemical approach might rank them according to the number of physical properties and reactions they share, whereas a mathematical approach might rank them according to their similarity in three dimensions.

The Daylight fingerprints described above effectively encode the substructures present in a molecule. It would not seem unreasonable that the proportion of substructures in common between two molecules should be a reasonable measure of similarity of the overall molecules. In mathematical terms this is a comparison of the bits in the fingerprints which are set on.

Note that the similarity measures are independent of the molecular feature descriptors. Fingerprints generated from structural keys or any other method can be used in the Daylight toolkits. The only requirements are that the size is a power of 2 and a multiple of 8. This ensures folding and translation to ascii representation works correctly. The current size limitations are 32 to 2^{30}

If we describe our molecules by the presence or absence of features, then the *binary association coefficients* or *similarity measures* are based on the four terms **a**, **b**, **c**, **d** shown in the two way table.

		OBJECT B		
		0	**1**	**Totals**
OBJECT A	**0**	**d**	**b**	**b+ d**
	1	**a**	**c**	**a + c = A**
	Totals	**a + d**	**b + c = B**	**n**

Where:
a is the count of bits on in object A but not in object B.
b is the count of bits on in object B but not in object A.
c is the count of the bits on in both object A and object B.
d is the count of the bits off in both object A and object B.

In addition:
n = (**a** + **b** + **c** + **d**)
A = (**a** + **c**)
B = (**b** + **c**)

Where:
n is the total number of bits on or off in objects A ***or*** B.
A is the count of the bits on in object A.
B is the count of the bits on in object B.

N.B. This nomenclature differs from that used by others, in particular the Sheffield group. In their system the labels **a** and **c** are reversed.

Pre 4.5 releases of the Daylight software provided an example of each of these classes.

- The **Euclidian** coefficient, **E** is defined as the square root of the ratio:

  (c + d)/n

  As both numerator and denominator are functions of **d**, **E** belongs to the first class of association coefficients dependent on the double zeros.

  As we do not take the square root, given mostly we are only interested in ranking ; it is simply a matching coefficient (*Sokal, R.R., Michener, C.D., (1958) The University of Kansas Scientific Bulletin **38**, 1409-1438*). In reality the value returned for this coefficient is the complement of this, i.e. the Hamming distance, or the Total Difference Coefficient (*Sneath, P.H.A., (1968) Journal of General Microbiology **54**, 1-11*)

  (a + b)/n

- The **Tanimoto** coefficient, **T**, is defined as the ratio:

  c/(a + b + c)

T is independent of **d**, i.e. **T** belongs to the second class of association coefficients defined above. It may be regarded as the proportion of the "on-bits" which are shared. See *Tanimoto, T.T. (1957) IBM Internal Report 17th Nov* see also *Jaccard, P. (1901) Bulletin del la Société Vaudoisedes Sciences Naturelles **37**, 241-272* .

N.B. These association coefficients are not necessarily true metrics. In particular those where the divisor depends on the particular pairwise comparison i.e. is not equal to **n**, may violate the triangle inequality. See *Anderberg, MR (1973) Cluster Analysis For Applications, Academic Press p 117*, for further discussion.

Over the years there has been much discussion as to which type of coefficient to use. In chemistry it has generally been thought that, as most descriptor features are absent in most molecules, i.e. the bit string descriptors such as the Daylight fingerprint contains mainly zeros, coefficients such as the Tanimoto are more appropriate. Further, given that the size of a Daylight fingerprint can be arbitrarily doubled, thereby adding mainly random off bits, any measure using matching off-bits, **d**, would be inappropriate. However this may not be the case for fixed width key based fingerprints. Daylight therefore offers access to both types of measure. The user must ensure that an appropriate one is chosen.

In version 4.5 Daylight extended the range of coefficients which could be used by introducing the asymmetric Tversky index. This allowed users to make use of directional similarity and harness the power of the concepts of prototypes in similarity searching.

As has been indicated above all of these indices are not monotonic, and as early as 1982 Hubalek (see *Hubalek, Z. Biol. Rev. (1982) **57**, 669-689)* showed that the coefficients could be clustered on the ranking of a given set of objects.

Recently Holliday *et al* (*Holliday, JD., Hu, C-Y. and Willett, P. (2002) Combinatorial Chemistry and High Throughput Screening **5,** 155-166*) have shown that a whole range of similarity measures can be clustered on the ranking of chemical structures.

With the release of version 4.9 therefore, we have introduced a whole range of named similarity measures and additionally allowed users to construct their own, from the terms **a**, **b**, **c, d** as appropriate.

6.3.1 Tversky Index

As of Release 4.51 the Tversky index (*Tversky, A. Psychological Reviews (1977)84 (4) 327-352*), provides a more powerful method for structural similarity searching. Its use and interpretation, however, are not not as simple as the Tanimoto index. The Tversky comparison is intrinsically asymmetric. As with Tanimoto the features present in two objects are compared. In the Tversky approach we have the concept of a "prototype" *to which* the objects or "variants" are compared. Note this differs from the Tanimoto index in which the similarity *between* two objects is estimated. This inherent asymmetry means that the Tversky index is very definitely not a metric. The *ratio model* in which the value is bounded (between 0 and 1) is defined as follows:

$$c/(\ *\ a +\ *\ b + c)$$

Setting the weighting of prototype features to the same value does not use the power of this index. Indeed, setting = = 1, produces the Tanimoto index. If = = 0.5 we get the Dice index. See below. The value of the index comes from setting the weighting of prototype and variant features asymmetrically, producing a similarity measure in a more-substructural or more-superstructural sense or reflecting the increased knowledge the user has about the prototype. Quite often one is looking for compounds *like* a known compound with appropriate properties.

Setting the weighting of prototype features to 100% (=1) and variant features to 0% (=0) means that only the prototype features are important, i.e., this produces a "superstucture-likeness" measure. In this case, a Tversky similarity value of 1.0 means that all prototype features are represented in the variant, 0.0 that none are.

Conversely, setting the weights to 0% prototype (=0) / 100% variant (=1) produces a "substructure-likeness" measure, where completely embedded structures have a 1.0 value and "near-substructures" have values near 1.0. Note: with no weight at all given to variant features, this measure is pretty sensitive to "noise" in Daylight fingerprints and settings of 90%/10% generally produce a more useful ranking.

Tversky measures where the two weightings add up to 100% (1.0) are of special interest. In XVMerlin the Tversky search query panel provides a "Sum 100%" checkbox which, when selected, forces the two weights to add up to 100%.

Advanced users may wish to experiment with Tversky indices where weightings are not limited to 100%. Weightings greater than 100% causes the distinguishing features **a, b** to be emphasized more than common features, **c**, which may be useful in analysis of diversity or dissimilarity.

6.3.2 User-defined and Named Similarity indexes

With the 4.9 release, users are able to use their own similarity measure throughout the software, where before they have been restricted to the hardcoded measures described above. The measure is entered as a string representing a **f(a, b, c, d)** with all of the usual mathematical expressions available. There is no restriction on the form of the function, but to work as a measure of similarity it should fulfill certain requirements. Expressions such as:

(c + d) /(a + b + c)
2.0*(c +d)/(a + b + 2.0*c)
(c + d)/(2.0* (a + b) + c)

are not useful, even though the user has attempted to alter the weights of the matched/unmatched pairs. It is nonsense to include in the numerator that which has been specifically excluded in the denominator (*Anderberg, MR (1973) Cluster Analysis For Applications, Academic Press p. 89*).

Users should also be cognizant of the restrictions imposed by the descriptors being compared. Daylight fingerprints can be arbitrarily doubled in size, as described above. The value of **d**, can be increased enormously without scaled increases in the information rich on-bits. It is not recommended therefore that indexes based on a function which uses the value of **d**, are used with Daylight type fingerprints. In the case of fingerprints based on structure keys there may be no such restrictions.

When doing comparisons users should also be aware of the range of possible values the index can take. Most have the range 0.0, 1.0 but this is not mandatory.

The most common useful indexes have been collected by Holliday *et al* (*Holliday, JD., Hu, C-Y. and Willett, P. (2002) Combinatorial Chemistry and High Throughput Screening 5, 155-166*) These are shown in the table, and can be referred to, by name, in applications and toolkits calls which allow user defined similarity functions.

Measure	Range	Formula

Cosine	0.0,1.0	$\frac{c}{\sqrt{(a+c)*(b+c)}}$
Dice	0.0,1.0	$\frac{2.0*c}{(a+c)+(b+c)}$
Euclid	0.0,1.0	$\sqrt{\frac{c+d}{a+b+c+d}}$
Forbes	0.0,	$\frac{c*(a+b+c+d)}{(a+c)*(b+c)}$
Hamman	-1.0,1.0	$\frac{(c+d)-(a+b)}{a+b+c+d}$
Jaccard	0.0,1.0	$\frac{c}{a+b+c}$
Kulczynski	0.0,1.0	$0.5*\left(\frac{c}{a+c}+\frac{c}{b+c}\right)$
Manhattan	1.0,0.0	$\frac{(a+b)}{(a+b+c+d)}$
Matching	0.0,1.0	$\frac{c+d}{a+b+c+d}$
Pearson	-1.0,1.0	$\frac{(c*d)-(a*b)}{\sqrt{(a+c)*(b+c)*(a+d)*(b+d)}}$
Rogers-Tanimoto	0.0,1.0	$\frac{c+d}{(a+b)+(a+b+c+d)}$
Russell-Rao	0.0,1.0	$\frac{c}{a+b+c+d}$
Simpson	0.0,1.0	$\frac{c}{\min((a+c),(b+c))}$
Tanimoto	0.0,1.0	$\frac{c}{a+b+c}$
Yule	-1.0,1.0	$\frac{(c*d)-(a*b)}{(c*d)+(a*b)}$

Notes

- The Tanimoto and Jaccard indexes are the same.
- The Forbes index has no upper limit.
- The Manhattan index is a distance = 1.0 - Matching index
- The Kulczynski index is the mean of the individual substructure similarities
- The Simpson index is the best of the individual substructure similarities
- The Dice index is the ratio of the bits in common to the arithmetic mean of the number of on bits in the two items.
- The Cosine index is the ration of the bits in common to the geometric mean of the number of on bits in the two items.

7. THOR - Chemical Database System

THOR (THesaurus Oriented Retrieval) is a *chemical* database system. That is, it is specifically designed to store and retrieve chemical information efficiently. Because THOR has built-in knowledge about chemical graph theory, it achieves high performance when storing chemical information. And more importantly, because it is specifically designed to store chemical information, it stores and retrieves it in ways that make sense to the chemist.

THOR's "language" is the common language of chemists. Users can store information using whatever nomenclature is appropriate or convenient; THOR understands the relationships between various nomenclatures and can retrieve a compound's data using any name ("identifier") that is known for the compound. For example, one might at different times choose to use SMILES, Wiswesser Line Notation (WLN), CAS numbers, local ID's, common, trade, and formal names. THOR allows chemical information to be referenced by an unlimited number of imprecise synonyms (e.g. states, mixtures, isomers, congeners, conformations), and maintains the relationships between each name, what is known about it, and how it is related to other names.

THOR makes no assumptions about the type or nature of the data that is to be stored, other than that it is information about chemical entities. Users can freely define their own datatypes -- descriptions of the data that are to be stored -- so any type or quantity of data can be stored. The definition of a datatype does not in itself cause any space to be used in a THOR database; space is allocated only when actual data are stored. Thus, hundreds or thousands of different types of data can reside in a single database efficiently.

7.1 Hash Table

Computer disks are designed to hold large amounts of information at relatively low cost, but they are slow at random-access operations - a typical disk can only make a few hundred random accesses per second. Thus, a database system that uses disk-resident databases should minimize the number of disk accesses it makes.

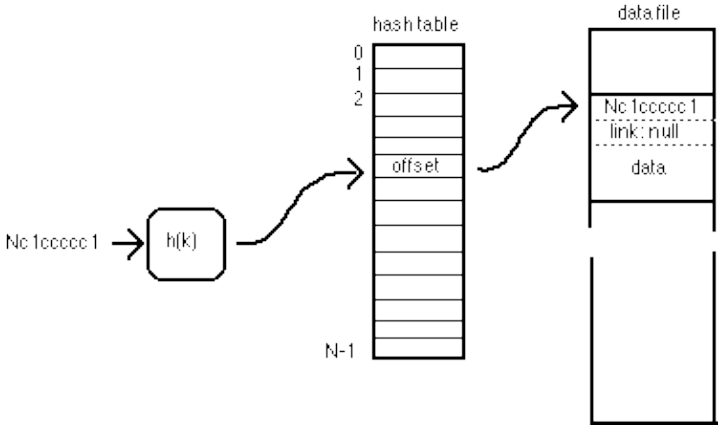
By using the unique SMILES of a molecule as the molecule's primary identifier (the TDT's "main topic"), THOR is able to eliminate all searching during data retrieval. All data are looked up directly in a *hash table*. This section describes, in an introductory fashion, how THOR's hash tables work.

Hashing begins with a *hash function*, h(K,N), which takes a string of characters, K, and converts it (via a pseudo-randomizing algorithm) into a number between zero and N-1. For example, given an identifier such as "Oc1ccccc1" that we want to find in the database, h("Oc1ccccc1",101) might produce the number 76.

The hash function has three noteworthy properties:

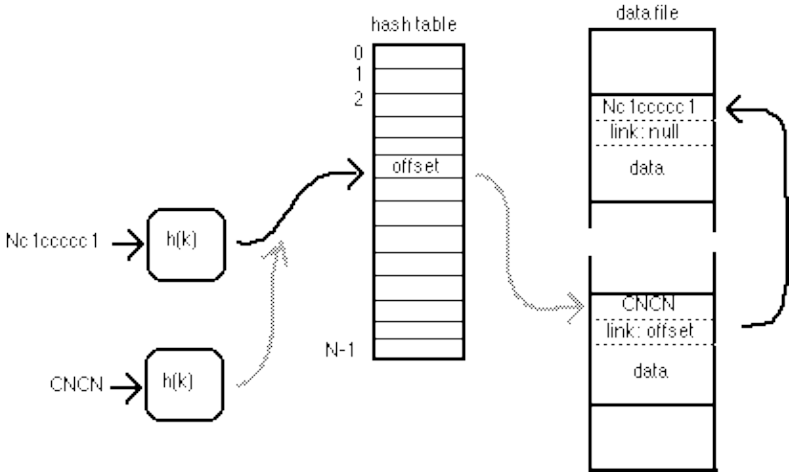
- Given a large number of keys K1-Kn, the integers I1-In produced by h(Ki,N) will be evenly distributed in the range 0 to N-1.
- The hash function can be computed very quickly.
- *Collisions* are possible, where two different keys Ki and Kj both produce the same result I.

Using a hash function h(K,N), data records on the computer's disk can be accessed directly: The hash value is used to access a *hash table*, which contains the desired record's location in the *data file*. Except in the case of hash collisions (discussed below), only two disk accesses are required (one if the hash table is *cached* discussed later in this chapter). The illustration below shows this operation.



Hashing to access a non-colliding record

In any hash-table system, *collisions* are possible: two keys Ki and Kj can hash to the same value (that is, h(Ki,N) equals h(Kj,N). When this happens, a *collision chain* is formed in the database (shown in the illustration below). The hash table always contains the location of the *head* record, and that record's *link* holds the location of the previous record with the same h(K,N), and so on to the last record in the chain, which has a *null* location to indicate that the end of the chain has been reached.



A collision-resolution chain for colliding records

Unless a database has zero or one records in it, there is a non-zero probability of collisions. As an analogy, imagine asking a random-number generator for 100 random numbers between 0 and 99. If the generator is truly random, one might expect roughly 70-80 unique numbers, and 20-30 duplicates. In a hash table where N=100, we can predict the same behavior when we store 100 records: there will be something like 20-30 collisions.

These collisions inevitably slow the data-access process down, since several disk accesses may be needed to find a record. In fact, at first glance one might guess that very long collision chains could form by random chance. Fortunately the odds are in our favor: The chances of very long chains forming are very remote, due to the statistics of random numbers.

More importantly, we can control these probabilities by setting the size of our hash table. If we know there are K records to be stored, a hash table with N=K/4 entries will give poor performance, one with N=K/2 entries will give reasonable performance, and one with N=K entries will give good performance.

THOR also has the ability to resize a hash table, because the data file and the hash table are completely separate, a hash table that is *overloaded* (K is much larger than N) can be discarded and a new, larger one built.

The most important feature of the hash-table mechanism is that data are accessed directly rather than by a time-consuming search. Hash tables exhibit what computer scientists call O(1) (pronounced "order one") performance: The access time of data doesn't depend on the size of the database. Looking at the examples above, we can see that the time required to look up any particular record is independent of how much data we have stored in the database. No matter how big the database becomes, the data access time only depends on three factors: how long it takes to compute h(K,N), how fast the disk is, and how many collisions we have on average. For a typical THOR system, h(k,N) takes a few microseconds, the average disk access is a few milliseconds, and if the hash table is properly sized, the average number of collisions is 1.2 to 1.5. Thus, access is roughly equal to the disk's access time.

The ability to use hash tables in THOR is directly related to the properties of SMILES. Unlike other identifiers used to name molecules, the unique SMILES is a fundamental property of the molecule, and there is only one unique SMILES for any particular molecule. This means that no matter how large a database grows, and no matter how many people enter data, all data for a particular molecule will be stored in the same record, and we can find that record by direct look-up rather than by searching.

7.2 Servers and Clients

The THOR system consists of two parts: servers and clients. The THOR server provides the ability to create, add data to, and read data from disk-based databases. The server allows many clients to read and write to the same database simultaneously, thus sharing the cost of the resource (disk) among many users. It can even provide database sharing across a wide geographical area.

In contrast, most of the complex aspects of THOR's operation, such as "standardizing" input file format, generating unique SMILES, merging input files, and so on, are handled at the client end. Many THOR client programs have a user interface of some sort. For example, the XVTHOR program uses X-Windows to provide a graphical display of data from a THOR database, and allows users to enter and edit chemical information; similarly, the 'sTHORman' program uses a "tty-style" user interface to manage THOR database. By contrast, the 'THORload' program has no user interface - it is "launched" and proceeds to load a database with no further user interaction.

7.3 Identifiers

THOR carefully distinguishes between identifiers and data. An identifier is something about which you have data, or about which you could have data. It is something to which you wish to attach information. As the name implies, it identifies.

THOR is unique in its ability to use a wide variety of different identifiers to access information, including imprecise synonyms, ambiguous names, isomeric and tautomeric names, 3D conformations, trade names, common names, formal names, and so forth. Some examples of common identifiers are SMILES, CAS number, company IDs, IUPAC name, and Wiswesser Line Notation (WLN). The choice of identifiers in THOR is up to the THOR manager; no restrictions are placed on what constitutes a proper identifier, but the "philosophy" of how to choose identifiers is important.

Frequently, identifiers are arbitrary names created by some convention or passed down through history. Examples of these arbitrary names are registrations numbers such as CAS, catalog numbers from vendors, and trivial names such as "morphine". Such identifiers carry no information about the chemical itself - the CAS number of 31604-28-1 is meaningless without a database. Likewise, "morphine" only has meaning because everyone knows what it is; the word itself has no chemical information.

But many identifiers do carry information about the chemical. Examples of this are IUPAC names, Wiswesser Line Notation (WLN), and SMILES; these all contain structural data. In other words, the name is both an identifier and contains chemical information. Since THOR is a database containing chemical information, we have to have a clear understanding about this dual role: that one thing can serve as both an identifier and as data.

It is rare for an identifier to name a single molecule with no possible variations. For example, 'dichlorobenzene' names three molecules: the meta-, ortho-, and para-substituted configurations; 'dichloroethylene' names the 1,1 as well as 1,2-cis and -trans configurations; the SMILES 'NC(CC)Cl' is a chiral structure with two possible configurations.

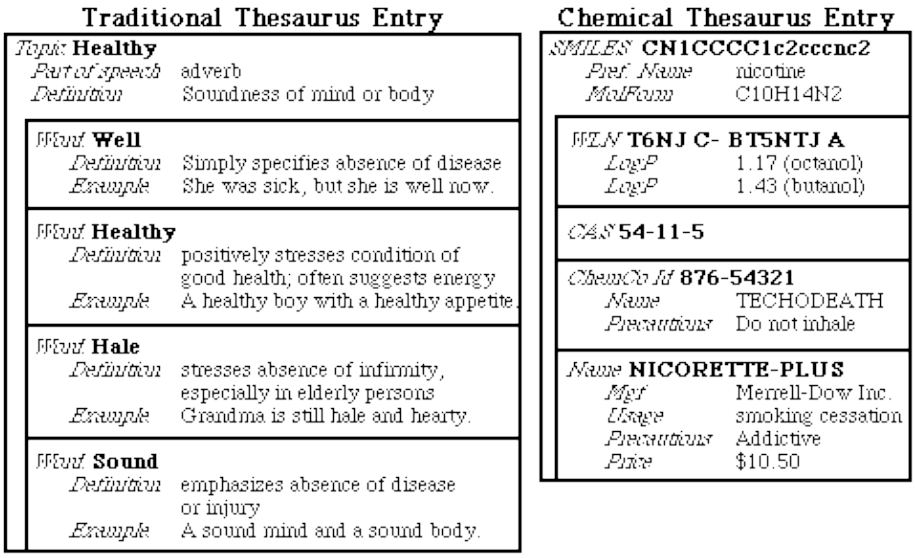
In contrast to the many other options, unique SMILES makes an ideal identifier for TDTs because of its special properties:

- Unlike many identifiers, SMILES represents a fundamental property of a molecule: Its bonds and atoms. Unlike common names, trade names, catalog numbers and company stock numbers, SMILES is not arbitrary. Rather, it represents information derived from the molecule itself. There is no arbitrariness about a SMILES.
- There no uncertainty about what is being represented; the meaning of each SMILES is unambiguous. For example, although ClC=CCl represents two molecules (cis- and trans-dichloroethene), it is clear that this is the case. Given any molecule, one can write down its SMILES; given any SMILES, one can tell if it represents a particular molecule. By contrast, although virtually every chemist would interpret "1,2-dichloroethene" to mean a mixture of the cis and trans configurations, there is nothing requiring this interpretation except convention.
- SMILES is understandable; most chemists learn basic SMILES in just a few minutes.
- SMILES can be read rapidly by a computer (this is not true of some line notations; for example it is very difficult to write a program to read Wiswesser Line Notation).
- Each molecule can be described by several SMILES, but the CANGEN algorithm will generate one special SMILES, the unique SMILES, for each molecule. The unique SMILES is universal: It is the same for every THOR database in the world. All THOR databases are intercompatible.

- The unique SMILES can be used to directly access THOR's records; no searching is necessary. This is discussed in more detail in the section on hash tables.

7.4 The THOR Data Tree

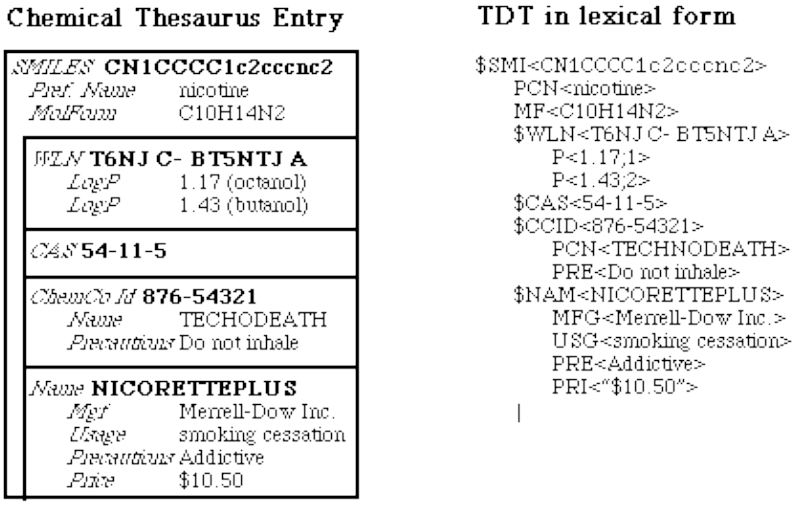
THOR uses a format with thesaurus orientation referred to as a THOR Data Tree or TDT in order to distinguish between identifiers and data associated with each identifier. To better understand this orientation, take a look at the traditional use of a thesaurus to capture relationships between a topic with one or more subtopics describing related ideas or concepts. Consider these two examples:



The above examples illustrate several important points about thesaurus entries:

- The thesaurus entries have a topic ("Healthy" and "CN1CCCC1c2cccnc2," respectively). All entries on the page are related to this topic.
- Sub-topics are not necessarily "subsets" of (completely encompassed by) the main topic. For example, the drug "NICORETTEPLUS" might include other active or inactive compounds not encompassed by the main topic.
- Each sub-topic has information (data) associated with it that may or may not pertain directly to the main topic, yet all of the topics are related in a fundamental way.
- Data can be attached to the main topic of the thesaurus entry as well as the sub-topics.
- Several of the sub-topics have alternate meanings. For example, "Sound" might also appear in a thesaurus entry under "Noise," a meaning entirely separate from the one used above.

The topics, sub-topics, and data shown in the right-hand example above constitute a TDT. Below is the lexical (string) form for the nicotine example above (note that the new-lines and indentation are ignored by THOR and are only for human readability).



The following concepts are critical to understanding the basic TDT format:

The *datafield* is the fundamental block of information in a TDT. A datafield is simply a string of printable characters. In the case above, examples of datafields are: `nicotine` and `C10H14N2`. If a datafield contains one or more of the characters "$<>;|", it must be quoted. For example: `"$10.50"`. Semicolons are used to separate individual datafields on a single line such as in `1.17;1` and `1.42;2`.

A *tag* is a label that names the type of data in the datafield. A tag consists of one or more alphanumeric characters. i.e., A-Z, a-z, and 0-9, plus the underscore character "_". Tags that start with a dollar sign indicate an identifier such as `$SMI` and `$CAS` in the example above. Tags that start with a slash ('/') are non-identifiers that are automatically cross-referenced to the tree root which may or may not be a SMILES. All other tags indicate items that are data only. For example: `PCN` and `MF`

A *dataitem* consists of a tag followed by angle-bracket-enclosed datafields. No space is allowed between the tag and the opening "<" Examples of dataitems are: `PCN<nicotine>` and `$CASPC<54-11-5>`.

A *datatype* is a set of definitions that indicate the meaning of a dataitem's fields. Each datatype is identified by its tag. See the section below for a more complete description and examples.

A complete *datatree* is represented as a series of dataitems with at least one identifier and terminated with a '|' (pipe).

Additional examples of TDTs are:

```
$SMI<CCC>MF<C3H8>$CAS<123-45-6>BP<123.4>$ISM<[13CH2]C>BP<124.3>|
$CAS<765-43-2>XCD<92.8;99.2;98.3;collected at STP>|
$SMI<c1ccccc1>$WLN<R>PCN<benzene>|
```

There is no restriction to the length of a TDT, the length of an identifier, the length of any dataitem, or the length of any datafield. THOR provides mechanisms for storing "binary data", data that can include anything at all. However, in order to take advantage of the datatree structure of a TDT, it must be rooted in $SMI.

Although, a TDT can be rooted in another identifier, it cannot contain mulitple branches.

7.5 Datatypes

A THOR datatype gives meaning to the data in a THOR database. For example, the data "$AID<1234-5> is meaningless without the definition of $AID, which tells us that it is an vendor catalog number. This is the primary purpose of datatype definitions.

A secondary function for datatype definitions is standardization of data and identifiers. Standardization is the process of modifying data according to particular instructions. For example, the datatype $NAM (name) is standardized by converting all characters to uppercase, eliminating all spaces and tabs, and removing punctuation. This greatly improves the likelihood that you will find what you are looking for. If, for example, you request "1,2-dichlorobenzene", THOR will retrieve a datatree that was entered even if it originally contained "1-2-DichloroBenzene"; both the stored name and your request will be converted to "12DICHLOROBENZENE". Note that some information is lost in this process, but carefully designed standardizations can be extremely helpful without sacrificing information.

7.5.1 Creating Datatypes

Datatype definitions are expressed as TDTs. This requires that there be at least a minimal set of predefined (truly universal) datatypes.

The following datatypes are the default datatypes:

$D<tag>	Internal tag for datatype
_V<vtag[;vtag...]>	Name for label
_B<btag[;btag...]	Brief name for pull-down menu
_N<ntype[;ntype...]>	Normalization parameters
_P<[*][;[*]...]>	Merlin in-memory pool inclusion flag --- * for all or if number N, then at most N datafields will be loaded of that type; alternatively, a '!' creates a row in Merlin from each subtree rooted by the identifier datatype to which it applies
_S<summary>	One-line summary of datatype's meaning and use
_D<description>	Long description of datatype's meaning and use.
_M<set>	Membership of the datatype for pull-down submenus
_C	General comments
_O	Owner of the datatype

Normally, a dataitem has a fixed number of fields. However, since these special datatypes are used to define the datafields of other datatypes, the _V specification can contain any number of datafields (each of which defines a datafield). Exactly one _V specification must occur in each datatype definition; it also sets the maximum number of datafields that may appear in the _N, _B, and _P specifications.

Here is a simple example that defines a SMILES datatype:

```
$D<"$SMI">
        _V<SMILES>
        _B<SMILES>
        _N<USMILES AUTOGEN $GRF>
```

```
        _P<*>
        _S<SMILES, primary identifier in the Daylight system>
        _D<SMILES is the primary key to all data in THOR system>
        _M<IDENTIFIER,SYSTEM,POOL>
        _C<SMILES, the fundamental identifier in THOR databases>
        _O<daylight@daylight.com (Daylight CIS, Inc., Irvine, CA>|
```

A more complex example is the definition of CLOGP, the computed partition coefficient of a compound. It has three fields in its definition (the real CLOGP is more complex):

```
$D<CP>
        _V<"CLOGP;#ERROR LEV;VERSION">
        _B<"clogp;clogp/err;clogp/ver">
        _P<"*;*;*">
        _N<"REAL32;INDIRECT $I;">
        _S<Estimate of the LogP(o/w) coefficient by CLOGP3>
        _D<Estimate of the partition coefficient LogP(o/w)
        computed by the CLOGP3 algorithm (from the MedChem
        Project)>
        _M<MODEL RESULTS,MEDCHEM,PHYSICAL PROPERTY,POOL>|
```

7.5.2 Standardization

The standardizations below work in a straightforward fashion: They modify the string representation of a single field in a straightforward fashion. During standardization of a TDT, if the THOR system discovers that the root of the TDT is not a SMILES, it will search the TDT for any datafield with a SMILES normalization. If one is found, it will use it to create a SMILES root for the TDT, "demoting" the original root to a subtree.

```
USMILES -- generate unique SMILES
ASMILES -- generate absolute SMILES (unique isomeric SMILES)
USMILESANY -- generate unique SMILES, unrelated to root
ASMILESANY -- generate absolute SMILES, unrelated to root
WHITE0, WHITE1, WHITE2 -- remove all, 2 or more, or 3 or more spaces,
respectively
UPCASE -- convert to upper case
DOWNCASE -- convert to lower case
NOPUNCT -- remove punctuation
CASNUM -- insert hyphens and verify checksum
INDIRECT -- designate indirect data field
INTEGER16, INTEGER32, REAL32, REAL64 -- designate numeric data format
BINARY -- denote binary data
SMILES_NTUPLE n -- designate SMILES-order n-tuple data where n is the
     n-tuple order; maintain order of data such as pairs of numbers
     for 2D coordinates (e.g. SMILES_NTUPLE 2) so that there is a
     one-to-one correspondence between the data and the molecule's atoms
     even after other normalizations like generating unique SMILES
PART_NTUPLE n -- designates component-order n-tuple data where n is the
number of data per part;
     maintain order for datatypes like FPP with a set of N fingerprints
     corresponding with N dot-disconnected SMILES representing a mixture
or library
```

```
AUTOGEN tag -- generate new dataitem using contents of dataitem specified
by 'tag'
GRAPH -- convert SMILES to GRAPH when retrieving data
MAKEGRAPH -- produce a GRAPH subtree of datatype $GRF
D3D -- compute 3D hash for use with $3D3 datatype
```

7.6 Database Anatomy

A THOR chemical database, usually thought of as a single entity, is actually made up of as many as three databases to store: datatypes, indirect references, and chemical information. By convention, these databases are referred to respectively as the *datatype-definition database*, the *indirect-data database*, and the *regular database*.

datatypes
: The *datatypes-definitions database* contains the definitions of all datatypes. Datatypes are frequently common to all databases at a particular site (i.e. all regular databases refer to the same datatypes database); this makes the data from all databases intercompatible.

indirect data
: The optional *indirect-data database* contains the expansions for indirect data. Like datatypes databases, indirect-data database are often shared by several related regular databases. Such indirect references save space when a field's contents is repeated many times in a database and allow uniformity by predefining a set of choices for users.

  Indirect datafields are defined by adding the INDIRECT specification to the normalizations. When the TDT is retrieved from the database, the indirect reference is looked up in the indirect-data database and the expansion data replaces the original indirect reference.

chemical data
: The *regular*, or *chemical-information database*, contains data about chemicals.

7.7 Database Files

Each database actually consists of six files; these are described below:

description
: The *description* file (suffix **`.THOR`**, also called the *header* file) describes the database. It contains the names of the other files, timestamps for each of the constituent files, the database's encrypted passwords, and the names of the databases where datatypes and indirect data can be found.

lockfile
: A *lockfile* (suffix **`.LCK`**) is present whenever a THOR server has the database open. It contains the process ID of the THOR server, and prevents two THOR servers from opening the same database.

primary data
: The *primary data file* (suffix **`.DP`**) contains all of the data in the database; this is where THOR datatrees are stored. A database can be completely rebuilt from the contents of this file.

primary hash

The *primary hash file* (suffix **.HP**) contains the hash table that allows "order one" (constant time) access of the primary data via SMILES.

cross ref.
The *cross-reference file* (suffix **.DX**) contains a cross-reference for each non-SMILES identifier; each non-SMILES identifier is listed with the SMILES of each TDT in which the non-SMILES identifier appears.

cross-ref. hash
The *cross-reference hash file* (**.HX**) contains the hash table that allows "order one" access of non-SMILES identifiers.

7.8 Reactions in THOR

Given that the THOR system uses SMILES strings for all of its database operations it works with reactions without modification. One can create a database with only molecules, a combination of molecules and reactions, or reactions only. It should be noted though that all reaction SMILES must be quoted within a datatree because of the ">" symbol used to separate the reactants and products within the SMILES string.

In many cases, combining molecule and reaction datatrees in a single database is the most appropriate organization for the database. When one builds a database, there is often a temptation to store molecule data (especially for the products) with the reaction. For example, physical data for reaction product(s) is commonly stored under the reaction. One perspective is that the physical data relates to the reaction because purification methods, solvents, etc. may affect the physical measurements and repeating the reaction multiple times will potentially give varying physical data. By the same token, the physical data can be considered to be about the product molecule and should be stored with it. The correct approach depends on the goals of the database and the types of queries which the end users wish to answer. Within the THOR system, both approaches are valid and may be employed.

There is one new and two extension of existing database standarizations that relate specifically to reaction processing.

```
MAKERXNMOL -- generate component molecules for a reaction;
     parses dataitems into dot-separated components
PART_NTUPLE n -- designate SMILES-order ntuple data over the
reaction components
SMILES_NTUPLE n -- designate SMILES-order ntuple data while
ignoring ">" symbols
```

8. Merlin - Exploratory Data Analysis Program

Merlin is a high-speed exploratory data analysis (EDA) program, designed to explore today's large chemical-information databases. Where THOR deals with TDTs one at a time, Merlin treats a database as a single unit and performs operations (e.g. searches and sorts) on the whole database. By using high-speed in-memory techniques, Merlin performs these EDA tasks with unprecedented speed.

In general, a user of an EDA program does not know exactly what information is required before starting work - the Merlin system is designed to allow information to be "discovered"

based on available data. EDA is similar to, but distinct from, data archival and retrieval. For example, a data archival/retrieval question might be, "What is the structure of Atromepine?" whereas EDA questions might be, "What known structures have names that are similar to Atromepine?" (e.g. Atropamine) and "What known compounds are structurally similar to Atromepine?" (e.g. Hyoscyamine).

Like the THOR system described in the previous section, Merlin provides access to a THOR database. However, the "view" Merlin gives of the database is quite different than THOR's view: THOR is a "microscope" for the database that provides a detailed view of individual datatrees, whereas Merlin might be thought of as a "macroscope" that performs operations on the database as a whole.

In the Daylight system, THOR provides data archival and retrieval services while Merlin provides exploratory analysis services. One advantage of using separate systems for archival and EDA is each can use the best strategies for the task at hand. This clear distinction differs from the design of most other systems in which a single methodology is used for all archival, retrieval, and search services.

8.1 In-Memory Searching

The basic idea behind Merlin is that data in a computer's main memory can be manipulated roughly five orders of magnitude faster than data on its disk. Throughout the history of computers, there has been a price-capacity-speed tradeoff for data storage: Large-capacity storage (tapes, drums, disks, CD-ROMS) is affordable but slow; high-speed storage ("core", RAM) is expensive but fast. Until recently, high-speed memory was so costly that even a modest amount of chemical information had to be stored on tapes or disks.

But technology has a way of overwhelming problems like this. The amount of chemical information is growing at an alarming rate, but the size of computer memories is growing even faster: at an exponential rate. In the mid-1980's it became possible for a moderately large minicomputer to fit a chemical database of several tens of thousands of structures into its memory. By the early 1990's, a desktop "workstation" could be purchased that could hold all of the known chemicals in the world (ca. 15 million structures) in its memory, along with a bit of information about each.

On the surface, in-memory operations seem like a straightforward good deal: A computer's memory is typically 10^5 times faster than its disk, so everything you could do on disk is 100000 times faster when you do it in memory. But these simple numbers, while impressive, don't capture the real differences between disk- and memory-based searches:

- With disk-based systems, you formulate a search carefully, because it can take minutes to days to get your answer back. With Merlin it is usually much faster to get the answer than it is to think up the question. This has a profound effect on user's attitudes towards the EDA system.

- In disk-based systems, you typically approach with a specific question, often a question of enough significance that you are willing to invest significant effort to find the answer. With Merlin, it is possible to "explore" the database in "real-time" - to poke around and see what is there. Searches are so fast that users adopt a whole new approach to exploratory data analysis.

There are three basic database operations in Merlin:

Searching: Merlin provides a number of search services. You can search for particular text or "ranges" of text (e.g. names, properties, activity, etc.), numeric ranges, similar molecular structures, substructures, and superstructures.

Sorting: Merlin can sort information using a variety of "comparison functions," including numeric, alphabetic, molecular formula, CAS number, etc.

Selecting: You can use several techniques to select items of interest "by hand."

Merlin presents the database as a "chemical spreadsheet" called a *pool*. The data are seen in *rows* and *columns*. The results of searches are stored in *hitlists*.

A **pool** is a THOR database that is loaded into the computer's memory.

A **row** is Merlin's representation of a THOR datatree. That is, all data in one row are from a single TDT in one database.

A **hitlist** is an ordered subset of the rows in the pool. Sorts and searches modify hitlists: A search adds or deletes rows from a hitlist, and a sort changes the order of the rows in a hitlist.

A **hit** is a row that is currently in a hitlist.

A **column** is a "vertical slice" through the pools, and contains data of one particular datatype. For example, a column might be for the datatype "Name," the column would contain a name from each row of the pool.

These concepts are discussed in more detail below.

8.2 Servers and Clients

The Merlin facility is organized in two parts: servers and clients. The Merlin server provides the basic capabilities in the Merlin system. The server creates and maintains all of the fundamental "objects" (pools, hitlists, and columns), and it carries out all of the main searching and sorting operations of the Merlin system. In addition, the server is entirely responsible for security: validating a user/password when a client connects, and validating database passwords when a database is open (security is discussed in more detail in the THOR_Merlin Administration Guide.

As with the THOR server, a Merlin server's *primary* purpose is to share resources. In THOR the resource being shared is access to disk-based databases, whereas in Merlin the resource being shared is the computer's memory. Merlin uses in-memory techniques for high-performance searching, so having sufficient memory and sufficient computer power is critical to Merlin's usefulness. A Merlin server allows many client programs to access the same data and computer, thus sharing the cost of the resources (memory and speed) among many users.

Although the Merlin server examines vast quantities of data during a typical search or sort operation, there is typically a relatively small amount of information that is communicated between the client and server. The client makes a request to the server, which might be a few

to a few hundred bytes of information. The server carries out the request (e.g. a sort or search), then typically only sends back those data that are to be displayed via the client's user interface. The Merlin system is designed so that the client/server communication is at a "low density" point in the "layers" of software, so that interprocess communication is minimized.

Merlin clients are primarily responsible for the "user interface" portion of the Merlin system's task. A typical Merlin client has a "Merlin Window" that showed the contents of a hitlist and some columns, allows a user to set up and carry out searches, and has a way of storing or printing the results.

Like the THOR system, there are many possible Merlin clients. The Merlin server is a program provided by Daylight, but the Daylight Toolkit provides access to all Merlin capabilities. Using the Toolkit, many possible clients can be created.

Two examples of Merlin clients are programs available from Daylight. The program XVMerlin is an X-Windows-based user interface to Merlin's search capabilities. The program 'sthorman' is a "tty-style" management tool used to load pools into Merlin, and to control security in the Merlin system.

8.3 Pools

A Merlin *pool* consists of data loaded into memory from a THOR database. To create a pool, the Merlin server reads the database's datatype definitions, the "_P" Merlin-pool-inclusion flag indicates which data are to be part of the pool, then copies data from the disk file to the computer's main memory, where they reside until the pool is unloaded.

8.4 Columns and Cells

Although a pool typically contains a subset of the data in a THOR database, it often consists of a wide variety of types of information, including structure, reactivity, chemical properties, prices, catalog numbers, and so forth. At any particular moment, the typical user is only interested in a few types of data. To solve this problem, Merlin provides *columns*.

A column of data is conceptually a "vertical slice" through all datatrees; it selects a specific datum or *derived-datum* (the result of computations on real data) from each datatree. The particular datum used for a column is defined by two properties:

Datatype
: When creating a column, you specify the datatype and the field within that datatype.

Function
: A datatree may contain several of a particular datatype (for example, there may be many names). Several *functions* are available to select from among those available, including "first", "last", "average" (for numeric data), "least", "greatest", and so forth; these are explained in detail below.

The pool can be thought of as a "chemical spreadsheet," with one row for each datatree in the database and various columns for different types of data. The intersection of a row and a column is called a *cell* and is the basic unit, or datum, in Merlin. Note: Setting the pool inclusion flag (_P) to '!' separates subtree data for "distinguished identifiers" into their own rows. This does not affect how the information is stored, only how it is viewed.

A *derived-data column* is a special type of column whose data are not in the database, but are derived from the database or computed during operations on the database. For example, a SIMILARITY column contains data computed when you perform a structural-similarity search. There is no SIMILARITY data in the database itself; the data are created and changed as you work. Other types of derived-data columns are discussed in the section below.

8.5 Column-creation Functions

As discussed above, *column-creation functions*, or simply *functions*, allow you to specify which instance of a particular datatype to use when more than one occurs in a row of the pool. The functions are:

First
: Use the first instance of the datafield in the row.

Last
: Use the last instance of the datafield in the row.

Least
: Use the lowest-valued instance of the datafield. For ASCII data, this is the lowest lexical value; for numeric data it is the lowest numeric value.

Greatest
: Use the greatest-valued instance of the datafield.

Longest
: Use the datafield that is the longest (contains the most characters). Not applicable to numeric data.

Shortest
: the datafield that is the shortest.

Count
: Creates a derived-data column containing the number of instances of the datafield.

Average
: Creates a derived-data column containing the average of all instances of the datafield. Only applies to numeric datatypes.

Standard Deviation
: Creates a derived-data column by computing the standard deviation of all instances of the specified datafield. Only applies to numeric datatypes.

8.6 Hitlists

A hitlist is an object that holds the results of sorts and searches. It is an ordered set of rows of data from a particular search pool. That is, it is a list of which rows are currently "hit" (selected), and the order in which you want the hits presented and operated on.

When a hitlist is first created, or after it is reset, it contains the set of all rows in the database in native order (native order is essentially random order, but it doesn't change as long as you have the pool open).

A hitlist's contents are changed by sorting operations (the hits are reordered) and by searching operations (rows are added to or deleted from the hitlist). These and other hitlist operations are discussed below.

The Merlin server allows clients to have several hitlists active at one time. For example, Daylight's XVMerlin program uses three: a primary hitlist, an "undo" hitlist, and a hitlist "memory". This allows users to undo mistakes, and to save a particular hitlist for later recall.