JOURNAL OF CHEMICAL INFORMATION AND MODELING

ARTICLE

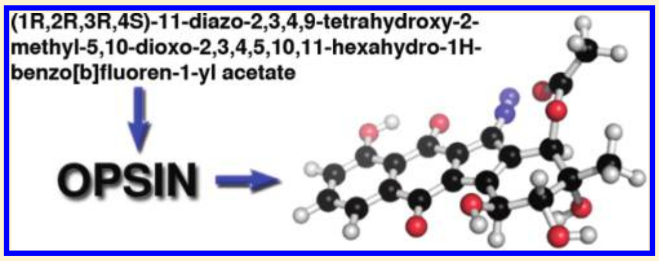
pubs.acs.org/jcim

Chemical Name to Structure: OPSIN, an Open Source Solution

Daniel M. Lowe, Peter T. Corbett, Peter Murray-Rust,* and Robert C. Glen

Unilever Centre for Molecular Science Informatics, Department of Chemistry, University of Cambridge, Lensfield Road, Cambridge, CB2 1EW, England

S Supporting Information


(1R,2R,3R,4S)-11-diazo-2,3,4,9-tetrahydroxy-2-methyl-5,10-dioxo-2,3,4,5,10,11-hexahydro-1H-benzo[b]fluoren-1-yl acetate
OPSIN

ABSTRACT: We have produced an open source, freely available, algorithm (Open Parser for Systematic IUPAC Nomenclature, OPSIN) that interprets the majority of organic chemical nomenclature in a fast and precise manner. This has been achieved using an approach based on a regular grammar. This grammar is used to guide tokenization, a potentially difficult problem in chemical names. From the parsed chemical name, an XML parse tree is constructed that is operated on in a stepwise manner until the structure has been reconstructed from the name. Results from OPSIN on various computer generated name/structure pair sets are presented. These show exceptionally high precision (99.8%+) and, when using general organic chemical nomenclature, high recall (98.7−99.2%). This software can serve as the basis for future open source developments of chemical name interpretation.

INTRODUCTION

With the ever increasing size of the scientific literature in the form of journal articles, patents, and theses, it is becoming infeasible for a scientist to manually view all the relevant literature. This has led to the creation of the field of chemical text mining to facilitate structure searching of the literature and to harness the power of computers to find novel relationships.[1] All such techniques rely on chemistry being presented in a computer-readable form. We deal here with the specific problem of extracting structural information from chemical names. A dictionary lookup approach is insufficient both because new compounds are created and because a single compound may often be named in so many different ways such as to make rigorous enumeration of naming variants impractical. Hence, chemical name to structure programs must be developed.

Research on Natural Language Processing (NLP) in chemistry began in the Unilever Centre in 2002 with tools to identify chemical concepts in natural language and to extract and interpret these concepts. This research led to creation of "the experimental data checker",[2] a tool for data parsing and validation. This evolved in 2003 into a component of OSCAR (Open Source Chemical Analysis Routines),[3] a tool for chemical entity recognition and resolution of identified chemicals to structures.

Our initial research into name to structure (V. de Souza, 2003) led to the creation of nesC (Name Enhancement System for Chemicals), a dictionary/lookup-based approach to the problem. This was followed by a solution that broke chemical names down into an XML parse tree (V. de Souza and P Murray-Rust, 2004) as part of the SciBorg project[4] that this was built on, leading to what is now called OPSIN (Open Parser for Systematic IUPAC Nomenclature).[5] This is employed as a name to structure resolver in OSCAR. This refactoring brought a major change from an ad hoc parsing system to one using regular grammar, paired with a lexer for tokenization. The project was subsequently further built upon and improved by one of us (DML, 2008) through changes in architecture, such as the replacement of the lexer and parser by a system that performs tokenization and parsing simultaneously. The resultant system includes support for considerably more chemical nomenclature leading to a greatly increased recall accompanied by increased precision and is the point of discussion of this paper.

Over many years, numerous recommendations for chemical nomenclature have been codified by the IUPAC (International Union of Pure and Applied Chemistry), IUBMB (International Union of Biochemistry and Molecular Biology), and CAS (Chemical Abstracts Service).[6−9] Despite their extensiveness, these recommendations are not comprehensive, often conflict with each other, and do not give a formal description of the grammar of chemical nomenclature. Nonetheless, these documents provide the best starting point for building a system to reverse chemical nomenclature.

At first glance, systematic chemical nomenclature appears to be sufficiently regular that it can be parsed by a grammar of the sort that is used for compiling computer programs. However, chemical nomenclature has the richness of a natural human language, in that it shows at least the following characteristics:

- There are several different approaches to representing chemical concepts that are in common use. Therefore, the system has to be able to use different vocabularies and different grammars to cope with these different styles of nomenclature.

Received: September 29, 2010
Published: March 09, 2011

Moreover, multiple styles of nomenclature may be present within the same name.

- The removal of characters (elision) and the addition of euphonic characters are commonly employed to yield better sounding but semantically identical names.
- The recommendations contain many exceptions and irregularities due to their development by many chemists in many countries over time.

Attempts to reverse chemical nomenclature into structures using computer programs date back to as early as 1962.[10] Solutions now exist from ACD/Laboratories,[11] Bio-Rad Laboratories,[12] CambridgeSoft,[13] ChemAxon,[14] ChemInnovation,[15] InfoChem,[16] and OpenEye.[17] However, these solutions are closed source, commercial, and with one exception have not published their workings. The exception is CambridgeSoft's Name=Struct publication[18] that employs a significantly different method of parsing chemical names.

OPSIN represents the first publication of an open, free, chemical nomenclature parser. It is written in Java allowing out of box support for the vast majority of platforms and is available as a standalone program or can easily be included as a library in a larger project.

Some key aspects of the program, which will be dealt with in more detail in the course of the paper, are listed below:

- Encoding and syntax: OPSIN is Unicode compliant and has a variety of conversions from conventional representations of non-ASCII characters, e.g., Greek letters into a standard normalized representation.
- Parsing: OPSIN has no initial preconception of how a chemical name should be broken down into words and further into tokens. A token, in OPSIN, is a string of characters that typically corresponds to a morpheme in a chemical name. A morpheme is the smallest part of a name that has semantic meaning, e.g., 'eth' in 'ethene' which means a carbon chain of length 2. OPSIN's grammar is used to direct this tokenization, with the tokens being assigned meanings in the process. The usage of white space and other English language token separators, such as hyphens, is quite different in chemical names from the usage in English language itself. Hence, English NLP approaches are not applicable. The usage of these delimiters in chemical names is not always consistent, and some omissions or inclusions of separators may be regarded as errors by the IUPAC recommendations. OPSIN takes a pragmatic approach to this problem and can make sense of many minor chemical illiteracies.

The result of this process is used to decide whether a space is indicative of the start of a new word and as a result words may include spaces. This parsing process may be performed starting from the left- or the right-hand end of the chemical name.

- Determining word meaning: OPSIN develops the concepts of chemical "words". OPSIN's words are usually the smallest free-standing component of a name. The combination of words to describe a compound has different semantics depending on the words in question. This relationship between the words must be determined to give the intended chemical interpretation. Mixtures that are described using a single name are handled by this system without difficulty.
- Component processing and assembly of fragments: The resolution of chemical nomenclature is processed in a stepwise manner proceeding successively through less localized nomenclature until the components of the name have been created. These are then assembled taking into account considerations such as valency. Subsequently the interactions between the words are considered using the previously determined word relationships. Finally, stereochemistry is applied, at which point the final structure can be serialized.

OPSIN stores vocabulary in XML files allowing the program's vocabulary to be modified to suit the problem without any code modification. This is the easiest way to extend OPSIN; extra vocabulary can be added simply by adding new entries to the appropriate XML files. Similarly OPSIN's grammar and rules for determining word meaning in multiword names are stored in external XML files.

Because of the wide range of codified nomenclature, development on OPSIN has thus far focused primarily on general organic nomenclature and amino acid nomenclature, with an emphasis on precision, so that any unsupported nomenclature would result in no answer rather than an incorrect answer.

IMPLEMENTATION

OPSIN has a workflow-like architecture, with each stage sequentially adding extra information until at the end of the process the structure is deduced from the chemical name (Figure 1). The components are stateless, which simplifies processing names through the system concurrently. This workflow has been designed to be modular with the interaction between the components as small as possible. The following sections will give an overview of the components of the workflow.

Preprocessing. The first step in OPSIN's workflow is to normalize the input text. This includes normalizing the representation of Greek characters to the Romanized name for the character, e.g., 'α' or '$a' are converted to 'alpha', converting white space to single spaces, and converting any non-ASCII characters to an ASCII replacement provided that their intended meaning is understood, e.g., normalization of various hyphens to the standard hyphen. Additionally, 'sulph', the traditional English spelling, is replaced by 'sulf'.

Parser. Chemical nomenclature can be thought of as being an artificial language. However, unlike other artificial languages, such as programming languages, the morphemes of chemical names are often not clearly delimited. One approach to this problem, as utilized by Corbett and Murray-Rust,[5] is to create all possible tokenizations for a chemical name on the basis of the program's lexicon. However, as such an approach does not take into account context, and many tokenizations will ultimately be found to be incorrect. For example 'propan-2-ol' would be tokenized to ['prop', 'an', '-', '2-', 'ol'] and ['propa', 'n-', '2-', 'ol'], for which the latter is clearly wrong (the 'n-' here is equivalent to that found in '*n*-butane'). As all possible tokenizations must be generated, assuming the number of places where tokenization is ambiguous is *n*, and that each instance results in two possibilities, this approach leads to 2^n possible tokenizations. In longer systematic names, this can cause an impractically large number of tokenizations to be generated and for the tokenization processes to take an unacceptably long time. To rectify this situation, the solution we have arrived at combines tokenization with parsing, such that only grammatical tokenizations can be made.

Before this system is explained, the question of what is being tokenized must first be answered. OPSIN parses chemical names on a per word basis. "Word" in this context refers to the smallest

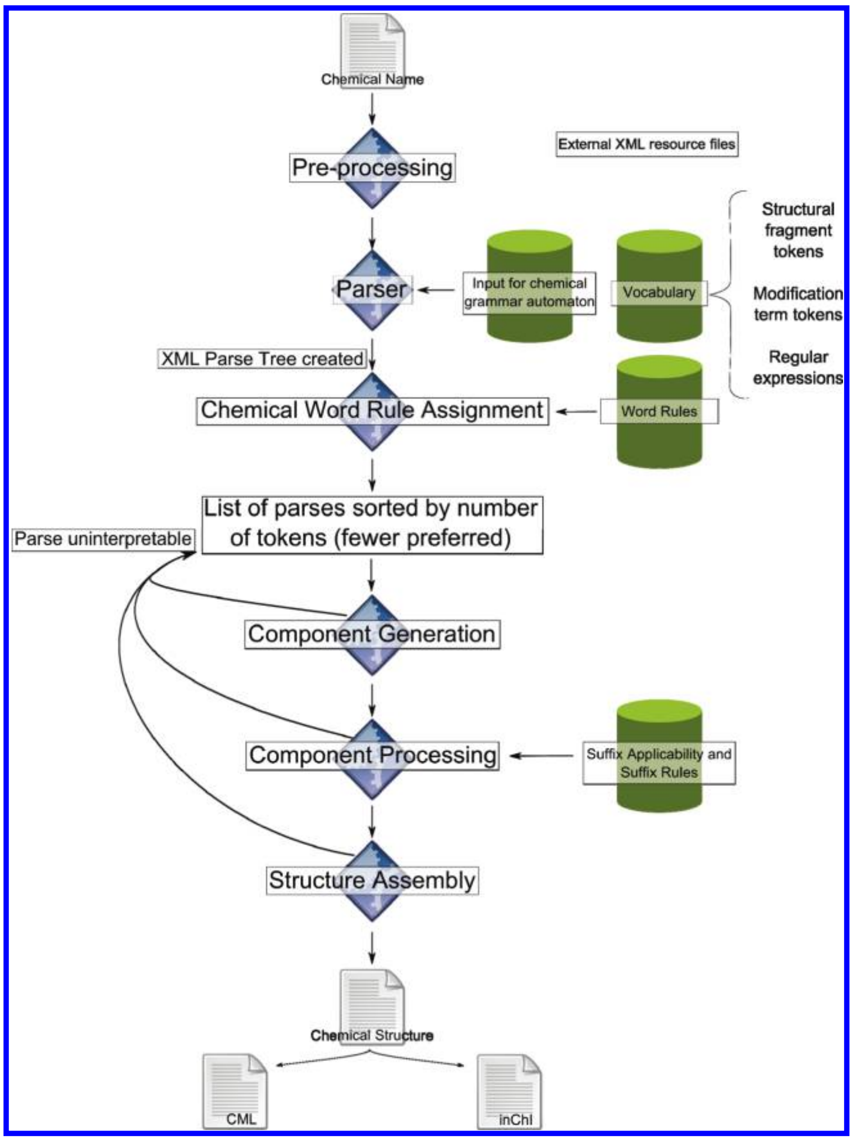


Figure 1. Components of OPSIN's workflow, showing the process from chemical name through to a structure.

Table 1. Examples of Chemical Names and the Number of Words in Each, As Determined by OPSIN

name	number of words
ethanoate	1
ethyl	1
ethylethanoate	1
ethyl ethanoate	2
acetic acid	1
acetic anhydride	2
vitamin c	1

meaningful unit of language. Table 1 gives a few examples of this. 'Vitamin C' is interpreted as one word because only when considered as a single unit does it have its intended meaning. Similarly 'acetic acid' is treated as one word because 'acid' on its own is not currently treated as being meaningful. Because our definition of a word is not defined by white space, it is the tokenization itself that will decide the word boundaries.

A chemical name can be thought of as corresponding to the following grammar:

```
<Chemical>::=<Word>+
<Word>::= <Substituent> | <Full> | <FunctionalTerm>
<Substituent>::=<Token>+
<Full>::=<Token>+
<FunctionalTerm>::=<Token>+
```

Where a "substituent" word describes a fragment of a chemical compound, e.g., 'ethyl', a "full" word describes a standalone chemical word, e.g., 'benzene' or 'ethylbenzene', and a "functional term" describes a modification term, e.g., 'ester'.

OPSIN's grammar, to date, describes 98 discrete classes of token. Each token class can either correspond to a list of tokens (e.g., 'benzen', 'pyridin', etc.) or, for classes that are not practical to enumerate, to a regular expression that describes all tokens of that class (e.g., an expression for a locant or for von Baeyer nomenclature). These lists of tokens and regular expressions are present in external XML resource files allowing the easy addition

Table 2. Meaning of the Token Characters in OPSIN's Grammar and Examples of to What They Can Correspond[a]

token character description	frequency	examples
locantGroup	28162	2, S-, alpha, N5
endOfSubstituent	25195	indicates the end of a substituent has been reached
inlineSuffix	14512	yl, ylidyne, oyl, sulfonyl
alkaneStem	12669	meth, eth, prop, tetracos
hyphen	12388	an optional hyphen
endOfMainGroup	9868	indicates the end of the principal group
simpleSubstituent	9529	chloro, hydroxy, amino
interSubstituentHyphen	9205	a hyphen between two substituents
simpleMultiplier	9015	di, tri, tetra
closeBracket	8979	], },)
openBracket	8979	[, {, (
ane	4028	ane as in the ending of an alkane or heteroatom analogue
trivialRing	3887	benzen, pyridin, toluen
stereochemistryBracket	3769	(2R), (2E,4Z)
suffixesThatCanBeModifiedByAPrefix	3578	amide, ate
suffix	2979	one, ol, carboxylic acid
o	2191	an optional euphonic o
unsaturator	2104	ene, en, yne
e	1964	an optional e
trivialRingSubstituentInlineOnly	1814	phen, imidaz, tol
bigCapitalH	1522	5H-
hydro	1168	hydro, dehydro
hwHeteroAtom	1022	aza, arsa, bisma
simpleGroup	918	hydroxide, chloroform, thiuram disulfide
acidStem	862	acet, valer, succin
inlineChargeSuffix	834	ium, ylium, ide, uide
heteroStem	818	alum, bor, oxid, sulf
aminoAcidEndsInIne	776	lys, alan, glutam
cyclo	758	cyclo as in cyclopropane
hantzschWidmanSuffix	721	iran, olan, inan
trivialRingSubstituentAnySuffix	653	pyrid, acrid
monoNuclearNonCarbonAcid	610	sulfam, azin, phosphon
fusionBracket	600	[4,5-d], [3′,4′:5,6]
ine	558	ine as in the ine of glycine
benzo	506	benzo as in benzo as a fused ring component
suffixPrefix	491	sulfon, sulfin, carbono
groupMultiplier	482	bis, tris, tetrakis
heteroAtom	433	aza, azonia, azanylia, azanida
implicitIc	414	added after unsuffixed amino acids to simplify systematic construction
fusionRing	403	indolo, pyrido, pyrrolo
endOfFunctionalGroup	394	indicates the end of a functional group has been reached
vonBaeyerMultiplier	339	bi, tri, tetra
vonBaeyer	263	cyclo[2.2.2]
standaloneMonovalentFunctionalGroup	260	chloride, cyanide
replacementInfix	230	thi, perox, hydrazid, hydrazon
alkaneStemModifier	159	iso, neo, tert
optionalCloseBracket	159	same as closeBracket but ignored after parsing
optionalOpenBracket	159	same as openBracket but ignored after parsing
simpleCyclicGroup	146	benzil
ringAssemblyMultiplier	137	bi, ter, quarter
locantThatNeedsBrackets	115	subset of locantGroup
chalcogenAcid	95	sulfon, sulfin, tellur
hwIne	94	ine as in the ending of a Hantzsch-Widman system
hwIneCompatible	94	oxa, thia, Selena

Table 2. Continued

token character description	frequency	examples
multiplyableFunctionalClass	94	oxime, oxide
multipleFusor	87	[2′,3′:3,4;2′′,3′′:6,7]
aminoAcidEndsInIc	78	glutam, aspart
FR2hydrocarbonComponent	76	cen, len, helicen
elidedAMultiplier	73	tetr, pent
infixableInlineSuffix	72	oyl
nonCarbonAcidNoAcyl	65	diphosphon, boron, selen
ringAssemblyLocant	52	2,2′:6′,2′′
cyclicUnsaturableHydrocarbon	51	menth, prism, adamant
lambdaConvention	47	3lambda5
groupStemAllowingInlineSuffixes	40	amid, keten, formazan
groupStemAllowingAllSuffixes	36	hydrazin
repeatableInlineSuffix	36	yl, ylidene
structuralCloseBracket	36	same as closeBracket but used to assist in nomenclature interpretation
structuralOpenBracket	36	same as openBracket but used to assist in nomenclature interpretation
divalentFunctionalGroup	31	ketone, sulfone
spiro	29	spiro as in a polycyclic spiro system
spiroLocant	29	subset of locantGroup used between components of a spiro system
suffixableSubstituent	29	sil, vin
hwAne	27	ane as in the ending of a Hantzsch-Widman system
hwAneCompatible	27	oxa, ox, thi
aminoAcidEndsInAn	26	tryptoph
annulen	23	[8]annulen
spiroDescriptor	17	spiro[2.2]
orthoMetaPara	16	ortho, meta, para, o-, m-, p-
cisOrTrans	15	cis, trans
anhydride	8	anhydride, peroxyanhydride
heteroAtomaElided	4	az, thi
aminoAcidYl	3	yl as in glycin-2-yl
basicFunctionalClass	1	ester, glycol, cyanohydrin
lightRotation	1	(+), (-), (+-)
acetalClass	0	acetal, ketal, hemiacetal, hemiketal
anhydrideLocant	0	1,2:3,4 (indicates groups to form anhydrides between)
chalcogenReplacement	0	thio, seleno, telluro
comma	0	s comma that is ignored after parsing
dispiroter	0	1,2′:7′,2′′-dispiroter
functionalModifier	0	poly
fusionRingAcceptsFrontLocants	0	naphthyridino, phenanthrolino
monovalentFunctionalGroup	0	alcohol, thiol
oMeaningYl	0	o as in glycino
perhydro	0	perhydro
relativeCisTrans	0	r-5,c-5,t-7
symPolycyclicSpiro	0	spirobi, spiroter
ylamine	0	ylamine (used in conjunctive nomenclature)

[a] The count is the number of times the token character was used in the first parse of 9601 successfully parsed IUPAC names from the ChEBI[20] database. It should be noted that many of these distinctions are purely to reject invalid interpretations of names; for example, all categories of heteroatom are treated identically after the parsing stage.

of new vocabulary. Individual tokens are associated in this XML with attributes containing semantic information, such as for ‘pyridin’ the structure of pyridine and for ‘tetra’ that its value is 4. Additionally, the type of element that will ultimately be created for this token when OPSIN produces its XML parse tree is indicated, e.g., “group” and “multiplier” for ‘pyridin’ and ‘tetra’, respectively.

The 98 token classes in the grammar are represented for convenience as single characters (which to avoid confusion with characters in the chemical name will be referred to as token characters) and are each associated with a short textual description of their meaning (Table 2). The grammar dictates which arrangements of these token characters are allowed. A regular grammar has some restrictions on the arrangements of token characters that can

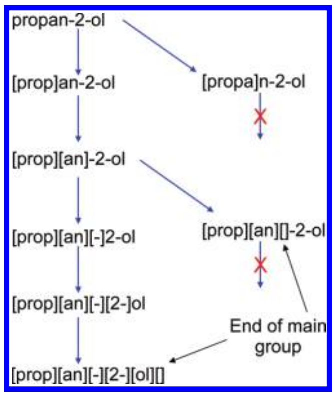


Figure 2. Example of how OPSIN's parser can quickly reject ungrammatical tokenizations. Note the paths through the automaton and how only one parse reaches an acceptable end point.

be defined, the importance of which will be discussed later. For a formal explanation of what constitutes a regular grammar, the reader is directed toward the work of Chomsky.[19]

The arrangements of the token characters are expressed as a large regular expression. This is then compiled into a deterministic finite-state automaton using the dk.brics.automaton package.[21] A deterministic finite-state automaton is formed of states, with each state having a set of allowed transitions. These transitions correspond to the set of token characters that the automaton may consume when in that state. Each transition leads the automaton to a new state. States that correspond to an acceptable end point are called "accept states". In OPSIN's grammar, these always correspond to a transition involving the endOfSubstituent, endOfMainGroup, or endOfFunctionalGroup token character.

To make maintaining and updating this regular expression tractable, it is expressed in terms of the descriptions of the grammar token characters, with aliases used for complex expressions. For example, there is an expression called "ringGroup" that describes any single ring, any von Baeyer ring system, or any trivial ring system. "ringGroup" is then used as part of the expression for ring assemblies, fused systems, and certain spiro systems. This regular expression alone is 839 characters, and the complete grammar is currently a 126,961 character long regular expression.

As previously mentioned, OPSIN does not treat white space as a hard delimiter; determination of what is considered a breaking white space and a white space that is part of a token is instead determined as a result of how the name has been tokenized. Tokenization and parsing occurs simultaneously as follows:

- From the current state in the discrete finite automaton, the list of allowed transitions is checked to determine which token characters are allowable next.
- For each allowed next token character, an attempt is made to match all corresponding tokens and regular expressions against the start of the chemical name.
- If a match is successful, the grammar token character and token is recorded, and this process is repeated with the state now being the state to which the transition led.
- The process terminates when no more of the name can be tokenized. The tokenizations that include the largest part of the name and end in an accept state are returned.

This process is performed iteratively and multiple routes may be found through the automaton yielding multiple parses. A parse is only successful if in addition to the requirement of ending in an accept state, the next character in the name is a white space or the end of the name (Figure 2).

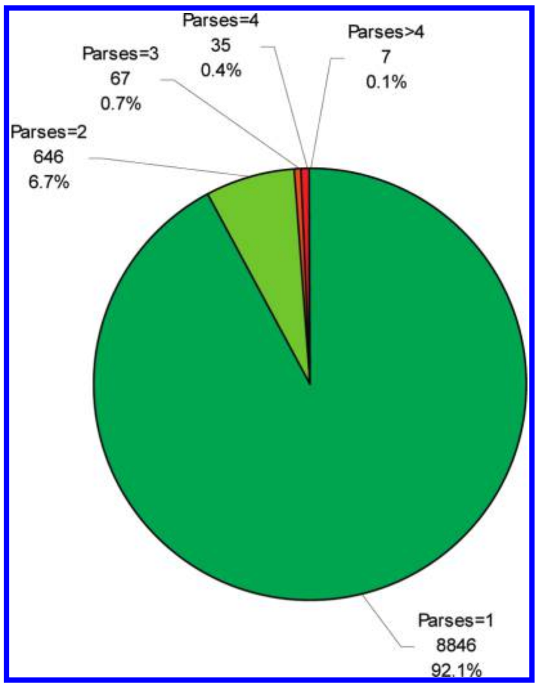


Figure 3. Number of parses generated from parsable IUPAC names in the ChEBI database. The vast majority of IUPAC names in the database produce only one or two parses. An example of a name in the four parses category was '3,7,11,15-tetramethylhexadeca-2,6,10,14-tetraen-1-yl diphosphate', for which the following parses were generated:

[3,7,11,15-, tetra, meth, yl, hexa, deca, -, 2,6,10,14-, tetra, en, -, 1-, yl] [di, phosphate]
[3,7,11,15-, tetra, meth, yl, hexa, deca, -, 2,6,10,14-, tetra, en, -, 1-, yl] [diphosphate]
[3,7,11,15-, tetra, meth, yl, hexadeca, -, 2,6,10,14-, tetra, en, -, 1-, yl] [di, phosphate]
[3,7,11,15-, tetra, meth, yl, hexadeca, -, 2,6,10,14-, tetra, en, -, 1-, yl] [diphosphate]

As an exception, space elision is allowed and subsequently corrected before functional terms, e.g., 'methylchloride' is converted back to 'methyl chloride' but to avoid misinterpretation is not currently performed between full terms, e.g., 'ethanehydrochloride' is not converted to 'ethane hydrochloride'.

To make this process as fast as possible when checking tokens against the start of the chemical name, only those that start with the first two characters are checked. Additionally, the regular expressions that correspond to the nonenumerable token classes are compiled in advance into deterministic finite-state automata to give optimal run time. The process of compiling regular expressions to deterministic finite-state automata can be quite slow, especially for the regular expression that describes the grammar; hence, the resultant deterministic finite-state automata are serialized and only updated if the regular expressions that generate them are altered.

After parsing has completed, all possible combinations of the parses for each word are then generated. For the majority of chemical names, this process results in only one complete parse for a chemical name (Figure 3).

Regular grammars are by definition unambiguous. However, as the grammar is being used to suggest to which token character the morphemes of a chemical name correspond, rather than the token character already being known, this assignment is not necessary unambiguous. Ambiguity can arise from different senses of a word, e.g., oxide can be a synonym for ether ('diethyl oxide') or mean the addition of oxygen ('trimethylphosphine oxide'). This can only be disambiguated in the next step, where the relationship between the words is considered. The other reason is that a term could exhibit ambiguity that is nontrivial to disambiguate. For example 'tetradecyl' is parsed as [tetradec][yl] or [tetra][dec][yl]. Cases of this type are rare and have been dealt with on a case by case basis, as part of the Component Generation component.

Grammar-based approaches for parsing chemical names have been described previously in the works of Kirby et al.[22−26] Their system utilized a context-free grammar based on their observation that all chemical nomenclature they had encountered, with the exception of enforcing the type of parentheses used, was possible in a context free grammar. They noted, however, that the majority of constructs in chemical names can be interpreted by a regular grammar. We also found this to be the case allowing OPSIN to parse chemical names using a regular grammar. Regular grammars are simpler than context free grammars, making them easier to write.

So far, the only significant drawback encountered in representing chemical nomenclature using a regular grammar has been the problem of representing bracketing. In a regular grammar, one cannot express (to an infinite depth) a language of the form `...((a))...`, where the number of open and close brackets is identical. It is, however, allowed to write the same expression but where the number of open and close brackets can be any number, i.e., not necessarily matched. Hence, OPSIN only matches opening and closing brackets with each other after the parsing stage.

OPSIN also supports parsing of CAS index names. These names are inverted such that the parent part of the name precedes its substituents, e.g., 'benzene, ethyl-'. To allow these to be parsed in the same way as normal chemical names, they must first be uninverted. The uninversion process can be achieved mostly by simply inspecting the components after the inversion comma for the presence or absence of an ending hyphen. For words without ending hyphens, it was found to be necessary to distinguish whether they were functional words or substituents which did not attach to the parent compound (e.g., those used in ester formation). This was achieved by using OPSIN's parser and observing which word type it assigned the word to be. As an example of this technique, OPSIN is able to recognize that the second word in 'disulfide, bis(2-chloroethyl)' is a substituent and hence rearrange the name to 'bis(2-chloroethyl) disulfide'. However, in the example '2(1H)-pyridinone, hydrazone, (2E)-', hydrazone is not moved to the front because it is a functional term, and it is rearranged correctly to '(2E)-2(1H)-pyridinone hydrazone'.

By default, OPSIN parses names from left to right, but by reversing the automata, it can also be used from right to left with near identical results. The number of states in the reversed chemical grammar automaton is significantly lower, 3388 states compared to 7750 in the left to right variant, indicating that there should be fewer routes through the automaton. This did not however translate into any improvement in tokenization speed. The ability to parse from right to left could potentially be useful in detecting exactly which part of an unparsable name is at fault and, hence, could potentially be useful in a future spell-checking algorithm. For example, in a name such as '1,3-dimethyl-4-unknownyl-benzene', OPSIN from left to right would be able to say '1,3-dimethyl-' was a substituent, and the name was parsable up to '1,3-dimethyl-4-'. From right to left, OPSIN would determine that 'benzene' was a full term and that the name was parsable up to 'yl-benzene'. Hence, this combination would single out the point of failure and identify "unknown(yl)" as a potential vocabulary term.

After parsing, an XML element is created for every token, except those that lack semantic meaning (e.g., an optional 'e' or an optional hyphen), to yield an XML parse tree. It should be noted that tokens from different token classes need not create different elements. For example, 'chloro' and 'meth' are tokens in different token classes, but both produce a "group" element. These elements become children of substituent, root, and functionalTerm elements with the special end of word grammar token characters being used to facilitate this chunking. These in turn are children of word elements. This is best illustrated with an example (Figure 4).

Chemical Word Rule Assignment. After parsing has been completed, a chemical name will have been tokenized into substituent, full, and functional words. "Word rules" describe the interactions between the words to OPSIN. For example, in 'ethyl ethanoate' (a substituent and a full word), the word rule 'ester' will be assigned, indicating that the ethyl group should be connected to the charged oxygen on the ethanoate with the charge removed. Without word rules, OPSIN would not know how the ethyl fragment and ethanoate group interact.

Word rule assignment is achieved by a mixture of looking at the string value of words, in particular the functional terms, e.g., 'ester', and looking in more detail at the XML OPSIN has generated for a particular word. The following are two examples of word rules employed by OPSIN:

```
<wordRule name="ester" type="full">
<word type="substituent" />
<word type="full" endsWithRegex="\S-(ate|amide|ite)[\]\)\}]*"/>
</wordRule>
<wordRule name="monovalentFunctionalGroup" type="full">
<word type="substituent" />
<word type="functionalTerm" functionalGroupType="monoValentStandaloneGroup"/>
</wordRule>
```

Additional word rules can be added trivially by adding entries such as the above to the appropriate XML file but must be backed up by code within the program, describing the operations that the word rule requires.

OPSIN's current set of word rules are illustrated in Table 3.

Our previously mentioned example name (Figure 4) after this stage will become

```
<molecule name="ethyl (1R,5S)-8-(chloromethyl)-8-azabicyclo-[3.2.1]oct-2-ene-3-carboxylate">
<wordRule type="full" wordRule="ester" value="ethyl (1R,5S)-8-(chloromethyl)-8-azabicyclo[3.2.1]oct-2-ene-3-carboxylate">
<word type="substituent" value="ethyl">
...
</word>
```

```
  <molecule name="ethyl (1R,5S)-8-(chloromethyl)-8-azabicyclo[3.2.1]oct-2-ene-3-carboxylate">
      <word type="substituent" value="ethyl">
          <substituent>
              <group value="2" valType="chain" usableAsAJoiner="yes" type="chain"
subType="alkaneStem">eth</group>
              <suffix value="yl" type="inline">yl</suffix>
          </substituent>
      </word>
      <word type="full" value="(1R,5S)-8-(chloromethyl)-8-azabicyclo[3.2.1]oct-2-ene-3-carboxylate">
          <substituent>
              <stereoChemistry type="stereochemistryBracket">(1R,5S)</stereoChemistry>
              <locant>8-</locant>
              <openbracket>(</openbracket>
              <group value="-Cl" labels="none" valType="SMILES" type="substituent"
subType="simpleSubstituent">chloro</group>
          </substituent>
          <substituent>
              <group value="1" valType="chain" usableAsAJoiner="yes" type="chain"
subType="alkaneStem">meth</group>
              <suffix value="yl" type="inline">yl</suffix>
              <closebracket>)</closebracket>
              <hyphen>-</hyphen>
          </substituent>
          <root>
              <locant>8-</locant>
              <heteroatom value="N" valType="SMILES">aza</heteroatom>
              <multiplier value="2" type="VonBaeyer">bi</multiplier>
              <vonBaeyer>cyclo[3.2.1]</vonBaeyer>
              <group value="8" valType="chain" usableAsAJoiner="yes" type="chain"
subType="alkaneStem">oct</group>
              <locant>2-</locant>
             <unsaturator value="2">ene</unsaturator>
             <locant>3-</locant>
             <suffix value="carboxylate" type="root">carboxylate</suffix>
         </root>
     </word>
 </molecule>
```

Figure 4. XML parse tree produced for 'ethyl (1R,5S)-8-(chloromethyl)-8-azabicyclo[3.2.1]oct-2-ene-3-carboxylate'. For this name, only one parse is produced.

```
  <word type="full" value="(1R,5S)-
8-(chloromethyl)-8-azabicyclo-
[3.2.1]oct-2-ene-3-carboxylate">
  ...
  </word>
  </wordRule>
  </molecule>
```

Word rules may be nested, allowing the interpretation of complicated names. For example, 'carbonyl cyanide m-chlorophenyl hydrazone' is first matched by the

Table 3. Word Rules and Example Names That Correspond to Them

word rule	example
acetal	propanal dimethyl acetal
additionCompound	carbon tetrachloride
acidHalideOrPseudoHalide	cyanic chloride
amide	nitrous amide
anhydride	acetic anhydride
carbonylDerivative	propanone oxime
divalentFunctionalGroup	diethyl ether
ester	ethyl ethanoate
functionalClassEster	acetic acid ethyl ester
functionGroupAsGroup	cyanide
glycol	ethylene glycol
glycolEther	ethylene glycol monomethyl ether
hydrazide	phosphoric hydrazide
monovalentFunctionalGroup	ethyl alcohol
multiEster	ethyl propyl methylphosphonate
oxide	thiophene 1,1-dioxide
polymer	poly(ethylene)
simple	ethylbenzene

monovalentFunctionalGroup word rule and then by the carbonylDerivative word rule.

If the molecule element has multiple wordRule children, this is indicative that the name describes either an ionic substance or a mixture. This would be true of a name such as 'benzene toluene'. For this name, benzene would be matched by the 'simple' word rule as would toluene, and this would then be interpreted as a 1:1 mixture of benzene and toluene. OPSIN can successfully interpret names of the form "compound (poly)hydrate" as a mixture of the compound with the given number of water molecules.

If no word rules match, a rule exists that allows substituents to be combined with other substituents or full words so that, for example, 'ethyl benzene' is interpreted initially as a substituent and a full word but then is converted to just one full word 'ethyl-benzene'. At the end of word rules assignment, all words should have been assigned to a word rule otherwise an error is thrown. Whether this error is thrown for names that correspond to just a substituent word (names formally representing radicals), e.g., 'ethyl', is controlled by a user-configurable switch.

Component Generation. If OPSIN has generated multiple parses, these will be sorted by the number of tokens into which the name was split, with fewer being preferred. The parses are evaluated sequentially, with the next parse only being attempted if processing the previous parse fails.

Each parse has been previously associated with an XML parse tree. This parse tree is successively modified throughout the remainder of the program as the nomenclature within it is processed. At this stage, self-contained nomenclature, e.g., annulenes, cyclizing alkanes, and von Baeyer nomenclature can be processed with a computer understandable representation of the resultant structures being added to the XML. These pieces of nomenclature can be thought of as conforming to microgrammars, each with their own rules for interpretation.

Brackets are matched at this stage, with the XML being appropriately modified such that the bracketed elements end up as children of bracket elements. Additionally at this stage, a few known cases of ambiguity are made unambiguous by rejection of the incorrect parse. A more in depth explanation of this and the following section is included in the Supporting Information.

By the end of this step, the structure of all components of the name will be known; that is, all "group" elements in the parse tree will contain a representation of the structure they describe. Many nomenclature operations can then be considering as modification of these components (e.g., the use of functional replacement to replace atoms) or as merging of these components (e.g., in fused ring nomenclature).

Component Processing. As Brecher[18] noted, the order in which chemical nomenclature is processed is important because many nomenclature operations can clearly be seen to rely on the completion of previous operations. For example, fusion of rings can only occur after the rings have been constructed. This component of the program is hence highly sequential. Some of the important tasks this component handles are the resolution of structures from a computer understandable form to an in memory connection table, the assignment of heteroatoms on Hantzsch-Widman rings, the forming of fused rings, the forming of ring assemblies, the forming of polycyclic spiro systems, assignment of locants, processing the effects of multipliers, and processing the effects of suffixes. An example of the XML parse tree after this stage is shown in Figure 5.

The computer understandable format used for input of structures is either SMILES[27] or CML[28] (Chemical Markup Language). These are stored along with the associated token and other useful attributes in external XML files. For example, the entry for aniline looks like the following:

```
<tokenList tagname="group" type="ring"
subType="arylGroup" symbol="z">
 ...
 <token value="Nc1ccccc1" labels="/1/2/
3/4/5/6" valType="SMILES">anilin</token>
 ...
```

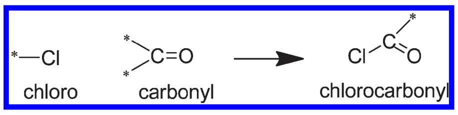
All other aryl group stems are present within this tokenList element. The symbol attribute corresponds to one of the 98 previously mentioned token characters in the chemical grammar (Table 2).

At the end of this step, all substituent and root elements will have only one group element, which is associated with a structural fragment.

Structure Assembly. At this stage the structural fragments produced in the previous step are connected together. This is performed in three steps that correspond to additive operations, locanted substitutive operations, and unlocanted substitutive operations, in that order. Each of these steps is performed on a word by word basis, in a right to left manner.

- Additive Operations

These involve the loss of a radical from both species, e.g., chlorocarbonyl.



Multiplicative nomenclature is also an additive operation, e.g., the joining of ethylene to the two nitrilo groups in ethylenedinitrilotetraacetic acid (Figure 6).

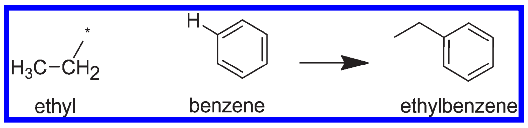
- Substitutive Operations

Substitutive operations are the most common in organic nomenclature and involve the loss of a radical from the substituent and

```
 <molecule name="ethyl (1R,5S)-8-(chloromethyl)-8-azabicyclo[3.2.1]oct-2-ene-3-carboxylate">
     <wordRule type="full" wordRule="ester" value="ethyl (1R,5S)-8-(chloromethyl)-8-
azabicyclo[3.2.1]oct-2-ene-3-carboxylate">
         <word type="substituent" value="ethyl">
             <substituent>
                 <group value="2" valType="chain" usableAsAJoiner="yes" type="chain"
subType="alkaneStem" defaultInID="2">eth</group>
                 <suffix value="yl" type="inline">yl</suffix>
             </substituent>
         </word>
         <word type="full" value="(1R,5S)-8-(chloromethyl)-8-azabicyclo[3.2.1]oct-2-ene-3-carboxylate">
             <bracket locant="8">
                 <stereoChemistry locant="5" value="S" type="RorS">5S</stereoChemistry>
                 <stereoChemistry locant="1" value="R" type="RorS">1R</stereoChemistry>
                 <substituent>
                     <group value="-Cl" labels="none" valType="SMILES" type="substituent"
subType="simpleSubstituent">chloro</group>
                 </substituent>
                 <substituent>
                     <group value="1" valType="chain" usableAsAJoiner="yes" type="chain"
subType="alkaneStem">meth</group>
                     <suffix value="yl" type="inline">yl</suffix>
                 </substituent>
                 <hyphen>-</hyphen>
             </bracket>
             <root>
                 <heteroatom value="N" valType="SMILES" locant="8">aza</heteroatom>
                 <group usableAsAJoiner="yes" type="ring" subType="alkaneStem" value="C1(CCCC2CC1)C2"
valType="SMILES" defaultInID="7">oct</group>
                 <unsaturator value="2" locant="2">ene</unsaturator>
                 <suffix value="carboxylate" type="root" locant="3">carboxylate</suffix>
             </root>
         </word>
     </wordRule>
 </molecule>
```

Figure 5. XML parse tree after all component processing has been performed. All the group elements are associated with the fragment of the structure they describe. The carboxylate suffix will have been attached to the bicyclic ring system. Compare with Figure 4 to clearly see the changes that have occurred.

the loss of hydrogen from the group to which it is attached.



Locanted substitution is processed before unlocanted substitution in the hope that this will allow the unlocanted substitution to be unambiguous. When performing unlocanted operations, the valency of atoms is taken into account, so that the resultant product will have an allowed valency. If this is not possible, no structure is returned.

Additionally during structure assembly, nondetachable features such as heteroatom replacement, indicated hydrogen, and unsaturation are applied. Again, this is first performed for locanted terms (which is performed just before additive

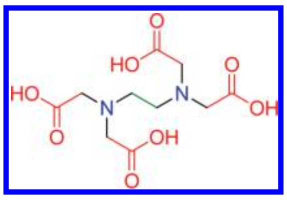


Figure 6. Ethylenedinitrilotetraacetic acid (ethylene, green; nitrilo, blue; acetic acid, red).

operations) and then for unlocanted terms (which is performed just before unlocanted substitutive operations).

The previously identified word rule is then used to indicate how the words will interact with each other. For example, if the word rule was anhydride, the acid word would be duplicated, an oxygen atom removed, and a bond formed to the duplicated acid.

Cyclic systems are then converted to Kekulé form. Previous to this point, cyclic systems with double bonds are represented as being single bonded with an indication that they are expected to form a double bond. For systems like pyrrolizine (Figure 7), all atoms will have this indicator even though when neutral, there is no possibility of all atoms having double bonds.

Indicated/added hydrogen and hydro are interpreted as cues to remove this indicator. Conversely, dehydro will add this indicator. When converting to Kekulé form, a final check that adding a double bond will keep the atom in an allowed valency is performed, otherwise, the indicator is removed. A check is then made that the number of atoms with the indicator is even, i.e., can an integer number of double bonds be added to satisfy all double bond creation? If this is not the case, either an atom that was saturated in the starting configuration of the fragment or a heuristically selected atom is saturated.

As a control on OPSIN's output, a valency check is performed on every atom. This is done by checking the current valency either against the stable valencies of the atom considering its element and charge or against that specified by the λ-convention valency of the atom[29] plus the change in the number of protons as indicated by charge modifying suffixes.

If the name describes a mixture or ionic compound, then a check is performed on whether the generated compound is neutrally charged overall. Where the stoichiometry is not explicit in the name and it is unambiguous as to which component of the mixture should have its stoichiometry adjusted, such as to yield an overall neutral component, that component will have its stoichiometry increased accordingly. For example magnesium(2+) chloride is initially treated as a 1:1 mixture but will have its stoichiometry adjusted to 1:2 to yield the intended neutral compound. Metals with unspecified charge are handled by being initially neutral and having their charge set to a value up to their maximum allowed positive charge.

Up to this point, hydrogen atoms have not been explicitly connected to any of the p-block atoms. By comparing the expected valency of the atom with the current sum of the order of its bonds, the number of hydrogens that would be expected to be there are calculated and added to the structure.

Stereochemistry, specifically *E*/*Z*/*R*/*S* and *cis*/*trans*, is then applied. This process starts by attempting to find all tetrahedral stereocenters and sp^2 stereocenters. This is performed using a derivative of the InChI canonicalization algorithm[30,31] to find atoms in the same environments. Tetrahedral atoms attached to four differently labeled atoms, meaning none of the attached

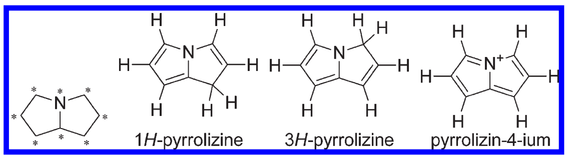


Figure 7. Left: Form OPSIN uses to store pyrrolizine. Right: Example structures that can be generated by setting the indicated hydrogen and charge.

atoms are equivalent, are identified as stereocenters. While this represents the majority of "true" stereocenters (cumulene stereocenters are not yet detected), it will not detect pseudoasymmetric centers,[32] that is those stereocenters which are dependent on the presence of other stereocenters. A good example of such a compound is '1,4-dimethylcyclohexane', which has two such stereocenters. Unlike the case of true stereocenters, the number of configurations is not necessarily 2^n; in the case of this molecule, there are only two configurations that are not degenerate. *Cis* and *trans* are interpreted as indicating the relative stereochemistry of substituents on pseudoplanar rings or, if such a system is not detected, are treated as synonymous with *Z* and *E*, respectively.

For each piece of stereochemistry in the name, once it has been determined to which of the detected stereocenters the stereochemistry corresponds, the chiral determinant must be determined. To do this, an implementation of the Cahn—Ingold—Prelog (CIP) rules[33−35] is used to order the atoms at the stereocenter in CIP order.

The resulting structure can be returned to the client either as CML or as InChI. Conversion to InChI is achieved using JNI—InChI[36] as an interface to the InChI library. CML has the advantage of being able to hold extra information, such as for each atom, all the locant values that have been associated with it.

Conversion to other formats, e.g., 2-D depiction, can be achieved by third-party software. For example, OPSIN is available as a Web service and demo site,[37] which utilizes the CDK[38] to generate a depiction and SMILES to supplement OPSIN's native CML and InChI output.

RESULTS

Testing the performance of name to structure conversion, while theoretically simple, in reality is impeded by the problem of finding a sufficiently large set of preannotated accurate name/structure pairs that is also representative of the chemical types of interest. The authors are not aware of an openly available gold standard set of name/structure relationships.

We consider here performance on three sets: a sample of 30,000 randomly chosen, Lexichem v2.0 generated, IUPAC names from the PubChem database; a set of 471,639 IUPAC names, generated by ACD/Name, retrieved from the ChemBridge catalogue;[39] and 16,055 IUPAC names retrieved from the ChEBI database that are primarily generated by ACD/Name with human curation in cases where the names do not conform to the latest IUPAC recommendations. The first two of these sets are primarily named using general organic nomenclature, while the ChEBI set also includes many names formed using more specialized biochemical nomenclature as well as inorganic nomenclature. For the ChEBI set, only entries that had a corresponding InChI, hence excluding radicals/polymers, were chosen. Additionally, entries that act as parents to other entries

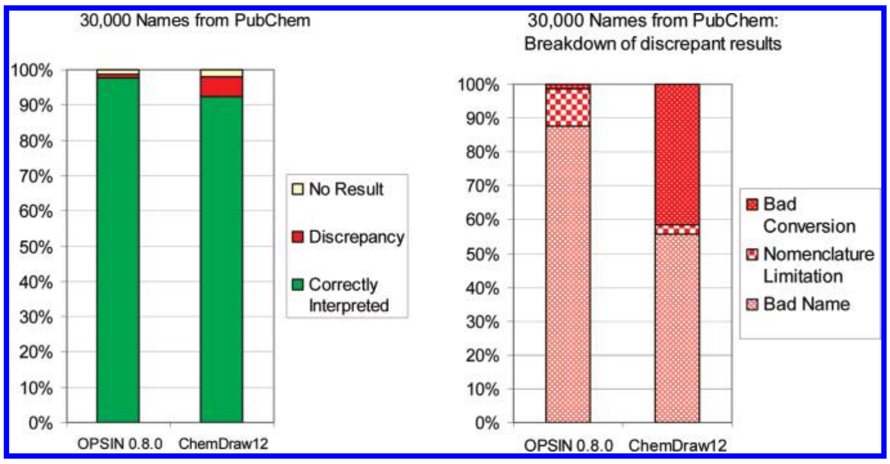


Figure 8. Results on the 30,000 names randomly selected from PubChem (a) and a breakdown of the reason for the discrepancy for names in the discrepancy category (b).

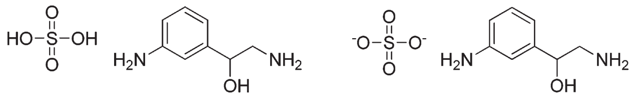


Figure 9. '2-amino-1-(3-aminophenyl)ethanol sulfate' as interpreted by OPSIN (left) and the structure present on PubChem, Compound ID: 6366791 (right).

such as alanine for L-alanine and D-alanine and, hence, describe multiple structures were excluded. While the PubChem and ChemBridge sets were unseen, names from an older version of the ChEBI database have previously been analyzed for agreement with their structures; hence, precision may be higher than otherwise expected on this set. All three sets are made available in the Supporting Information.

Checking the correctness of OPSIN's results was performed by comparing the InChIs that OPSIN generated with InChIs extracted from PubChem, ChEBI, or those generated from SDF file in the case of the ChemBridge catalogue. InChIs from PubChem and ChEBI were compared by checking the agreement of the main, charge, stereochemical, and isotopic layers. In the case of the ChemBridge set, checking of the stereochemical layer was not performed as stereochemistry was present in many of the structures, but there was a lack of stereochemical information in the majority of the names. A theoretical weakness of this approach is that it does not compare the exact tautomer. However, in previous testing, this was found to make a negligible difference, and the use of standard InChIs avoids any potentially lossy format conversions. For cases where OPSIN's InChI did not match, the reason for the mismatch was identified by manual inspection of the names and the result the program had produced. To give some context to OPSIN's performance, ChemDraw 12.0[40] was also tested under the same conditions. The representation of superscripted numbers and Greek letters proved problematic, especially on the ChemBridge set, and hence to represent ChemDraw in the best possible light, superscripts and Greek letters were, where possible, converted to a form that the program could understand. The data underlying the discrepant results charts are available in the Supporting Information.

On the PubChem set, Figure 8, the largest cause of error came from incorrectly or ambiguously named compounds. Note that in this analysis an ambiguous name that falls into the "Discrepancy" category will be subcategorized as a "Bad Name", but if the expected structure were generated from it, it would fall into the "Correctly Interpreted" category. The main causes of bad names were incorrect fused ring numbering, missing stereochemistry, and missing stoichiometry. Another source of discrepancy was in names that describe an overall charged mixture that is likely to be a salt (Figure 9). For names of this type there is a conflict between the different rules of the IUPAC recommendations as to how they should be interpreted. OPSIN implements rule C-816.4 of the IUPAC 1979 guidelines,[6] which is at odds with many other rules where names such as "sulfate" are reserved for describing the anion. As discrepancies on these names are due to ambiguities in IUPAC nomenclature, they form a distinct category referred to in the graphs as "Nomenclature Limitations". Because of the systematic nature of generated names, the errors in conversion by ChemDraw were caused by only a small number of problems, with one repeated mistake accounting for over half of the bad conversions. The mistake in question was the interpretation of (R-yl)methylideneamino, where R is any string, for which it is not reasonable for both the methylidene and the other substituent to be bonded to the amino due to amino only having two substitutable hydrogen. Despite the names being computer generated, it can be seen from panel (b) of Figure 8 that it is more often than not, especially in the case of OPSIN, the name that is at fault rather than the conversion. This also occurred on the other two test sets (Figures 10b and 11b).

OPSIN achieved near identical results to panel (a) of Figure 8 when presented with all 26 million Lexichem v2.0 IUPAC names from PubChem. Because of the program's speed, approximately 5.3 ms per name on a 3.4 GHz Pentium 4 to InChI (or 25% faster to CML), batch processing of such sets does not present a problem.

■ DISCUSSION

On the test sets analyzed in this publication, OPSIN has proved to be significantly more accurate than one of the leading commercial software solutions and offers comparable recall when processing IUPAC general organic nomenclature. OPSIN has been developed with extensive tests, including large regression sets based on known name to structure relationships, to allow the rapid spotting of any bugs introduced as new nomenclature is added and to draw attention to areas that should be improved.

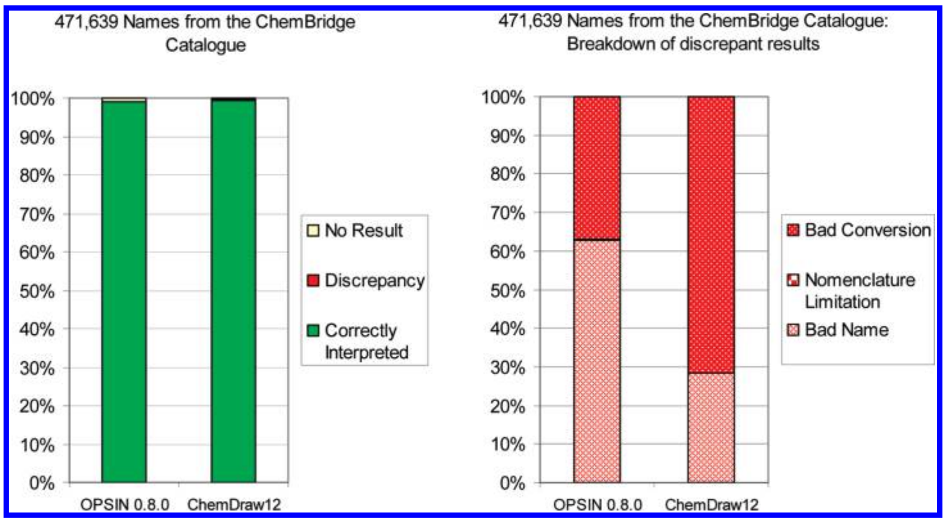


Figure 10. Results on the names extracted from the ChemBridge catalogue (a) and a breakdown of the reason for the discrepancy for names in the discrepancy category (b). The percentage of names that produced discrepant results was 0.13% and 0.35% for OPSIN and ChemDraw, respectively. Note that without modifying the representation of superscripts, ChemDraw's performance falls to 97% recall.

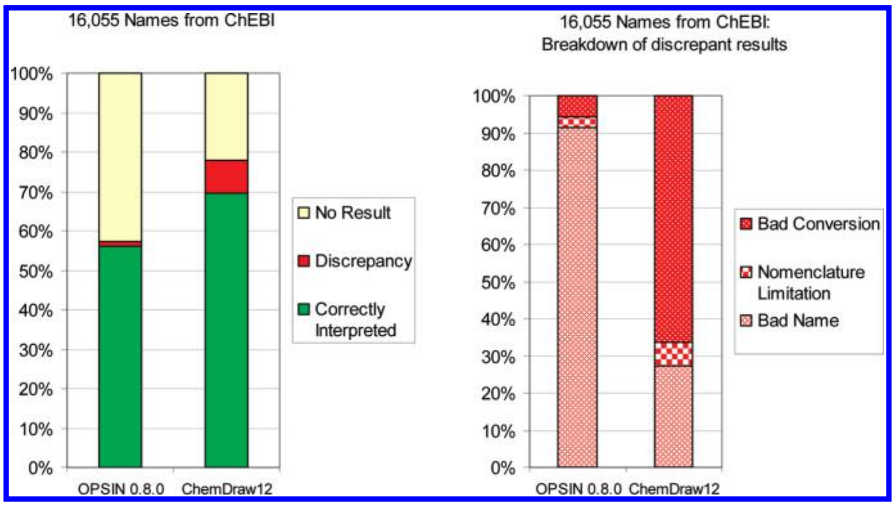


Figure 11. Results on the names from the ChEBI database (a) and a breakdown of the reason for the discrepancy for names in the discrepancy category (b). Unlike the other sets, a significant number of biochemical names, e.g., natural products, carbohydrates, and inorganic names, are included, resulting in OPSIN having significantly lower recall. It should be noted that the majority of ChemDraw's bad conversions actually arise from the biochemical and inorganic names that are as yet uninterpretable by OPSIN, although OPSIN's precision on the names that were interpretable by both is still significantly higher.

OPSIN is currently used via a Web service by the Chemistry Add-in for Word,[41] providing a useful way of rapidly converting textual descriptions to structures. OPSIN is also used by OSCAR3, an open source tool for identifying and marking up chemical terms.[42]

OPSIN has also been used with great success to detect name/structure mismatches in ChEBI, a curated database of molecules of biological structures with, on multiple occasions, the structure in the database being found to be incorrect. Over 100 corrections to names or structures have resulted from this collaboration.

Current Limitations and Future Work. While OPSIN covers the majority of general organic nomenclature, it cannot claim to comprehensively cover all of the IUPAC recommendations. Even if all recommendations could be encoded, there would still exist a large number of names that deviate significantly from IUPAC recommendations but are still unambiguous, e.g., dropping brackets from stereochemistry terms. It should be noted that with the exception of amino acid nomenclature, OPSIN does not yet support biochemical nomenclature, e.g., carbohydrate nomenclature, and hence will have very low recall when presented with such names (Figure 11). Support for inorganic nomenclature is also limited, partially as improved support rapidly leads to the problem of representing inorganic structures in a sensible way that can be understood by other cheminformatics packages. It is hoped that, as OPSIN is an open source project, the community will contribute to those areas for which there may be demand, yet OPSIN is lacking sufficient functionality.

In real world usage, the quality of chemical names in the form of typographical or OCR errors and malformed names will also present difficulties. OPSIN is tolerant of the majority of hyphenation and space omission/addition errors and is case insensitive, where doing so does not introduce ambiguity. Adding simple spelling corrections is a possible future improvement.

CONCLUSIONS

We have produced a comprehensive, fast, precise, and extensible chemical name to structure interpretation algorithm. Compared to the tested commercial offering, OPSIN was found to be significantly more precise, while still offering comparable recall on IUPAC general organic nomenclature. By employing a strict grammar, OPSIN can elegantly fail on chemical names that include nomenclature that is not yet supported. OPSIN is free to all, which we hope will lead to its development and widespread use in extracting chemical structural data from the chemical literature.

ASSOCIATED CONTENT

Supporting Information. Three sets of names with their corresponding InChIs, raw data underlying the discrepant result analyses, and a more extensive discussion of the processes that OPSIN goes through when converting the XML parse tree to a chemical structure. This material is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author
*pm286@cam.ac.uk.

ACKNOWLEDGMENT

D.M.L. thanks Boehringer Ingelheim for funding, Albina Absdulina for her contribution to fused ring nomenclature support, Sam Adams for coding advice, and other members of the Unilever Centre who have contributed to the project.

REFERENCES

(1) Banville, D. L. Mining chemical structural information from the drug literature. *Drug Discovery Today* **2006**, *11*, 35–42.
(2) Adams, S. E.; Goodman, J. M.; Kidd, R. J.; McNaught, A. D.; Murray-Rust, P.; Norton, F. R.; Townsend, J. A.; Waudby, C. A. Experimental data checker: Better information for organic chemists. *Org. Biomol. Chem.* **2004**, *2*, 3067.
(3) Townsend, J. A.; Adams, S. E.; Waudby, C. A.; de Souza, V. K.; Goodman, J. M.; Murray-Rust, P. Chemical documents: Machine understanding and automated information extraction. *Org. Biomol. Chem.* **2004**, *2*, 3294.
(4) Copestake, A.; Teufel, S.; Rupp, C.; Siddharthan, A.; Waldron, B.; Murray-Rust, P.; Corbett, P.; Parker, A.; Hayes, M. *SciBorg.* http://www.cl.cam.ac.uk/research/nl/sciborg/www/ (accessed January 12, 2011).
(5) Corbett, P.; Murray-Rust, P. High-throughput identification of chemistry in life science texts. *Lect. Notes Comput. Sci.* **2006**, *4216*, 107–118.
(6) IUPAC. *Nomenclature of Organic Chemistry*; Pergamon Press: Oxford, 1979.
(7) IUPAC. *A Guide to IUPAC Nomenclature of Organic Compounds (Recommendations 1993)*; Blackwell Scientific Publications: London, 1993.
(8) IUPAC. Draft Nomenclature of Organic Chemistry. http://old.iupac.org/reports/provisional/abstract04/favre_310305.html (accessed January 12, 2011).
(9) *Nomenclature of Inorganic Chemistry: IUPAC Recommendations 2005*; Royal Society of Chemistry Publishing/IUPAC: Cambridge, U.K., 2005.
(10) Garfield, E. An algorithm for translating chemical names to molecular formulas. *J. Chem. Doc.* **1962**, *2*, 177–179.
(11) *ACD/Name*; ACD/Labs: Toronto, Canada.
(12) *IUPAC DrawIt*; Bio-Rad Laboratories: Hercules, CA.
(13) *Struct=Name*; CambridgeSoft: Cambridge, MA.
(14) *Name to Structure*; ChemAxon: Budapest, Hungary.
(15) *NameExpert*; ChemInnovation Software: San Diego, CA.
(16) *Name to Structure*; InfoChem: Munich, Germany.
(17) *Lexichem ToolKit*; OpenEye Scientific Software: Santa Fe, NM.
(18) Brecher, J. Name=Struct: A practical approach to the sorry state of real-life chemical nomenclature. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 943–950.
(19) Chomsky, N. Three models for the description of language. *IRE Trans. Inf. Theory* **1956**, *2*, 113–124.
(20) Degtyarenko, K.; Matos, P.; de; Ennis, M.; Hastings, J.; Zbinden, M.; McNaught, A.; Alcantara, R.; Darsow, M.; Guedj, M.; Ashburner, M. ChEBI: A database and ontology for chemical entities of biological interest. *Nucleic Acids Res.* **2008**, *36*, D344–350.
(21) Møller, A. dk.brics.automaton: Finite-State Automata and Regular Expressions for Java. http://www.brics.dk/automaton/ (accessed January 12, 2011).
(22) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer translation of IUPAC systematic organic chemical nomenclature. 1. Introduction and background to a grammar-based approach. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 101–105.
(23) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer translation of IUPAC systematic organic chemical nomenclature. 2. Development of a formal grammar. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 106–112.
(24) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. Computer translation of IUPAC systematic organic chemical nomenclature. 3. Syntax analysis and semantic processing. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 112–118.
(25) Cooke-Fox, D. I.; Kirby, G. H.; Lord, M. R.; Rayner, J. D. Computer translation of IUPAC systematic organic chemical nomenclature. 4. Concise connection tables to structure diagrams. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 122–127.
(26) Cooke-Fox, D. I.; Kirby, G. H.; Lord, M. R.; Rayner, J. D. Computer translation of IUPAC systematic organic chemical nomenclature. 5. Steroid nomenclature. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 128–132.
(27) Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31–36.
(28) Murray-Rust, P.; Rzepa, H. S. Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 928–942.
(29) Powell, W. Treatment of variable valence in organic nomenclature (Lambda Convention). *Pure Appl. Chem.* **1984**, *56*, 769–778.
(30) Tchekhovskoi, D. InChI Canonicalization Algorithm. http://sourceforge.net/mailarchive/forum.php?thread_name=5.1.1.5.2.20050708111329.02502190%40email.nist.gov&forum_name=inchi-discuss (accessed January 12, 2011).
(31) InChI Technical Manual. http://www.iupac.org/inchi/ (accessed January 12, 2011).
(32) Razinger, M.; Balasubramanian, K.; Perdih, M.; Munk, M. E. Stereoisomer generation in computer-enhanced structure elucidation. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 812–825.
(33) Cahn, R. S.; Ingold, C.; Prelog, V. Specification of molecular chirality. *Angew. Chem., Int. Ed. Engl.* **1966**, *5*, 385–415.
(34) Prelog, V.; Helmchen, G. Basic principles of the CIP-system and proposals for a revision. *Ange. Chem., Int. Ed.* **1982**, *21*, 567–583.
(35) Mata, P.; Lobo, A. M.; Marshall, C.; Johnson, A. P. The CIP sequence rules: Analysis and proposal for a revision. *Tetrahedron: Asymmetry* **1993**, *4*, 657–668.
(36) Adams, S. JNI-InChI. http://jni-inchi.sourceforge.net/ (accessed January 12, 2011).
(37) OPSIN Web Interface. http://opsin.ch.cam.ac.uk/ (accessed January 12, 2011).
(38) Steinbeck, C.; Han, Y.; Kuhn, S.; Horlacher, O.; Luttmann, E.; Willighagen, E. The Chemistry Development Kit (CDK): An open-source

java library for chemo- and bioinformatics. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 493–500.

(39) ChemBridge. http://www.chembridge.com/ (accessed January 12, 2011).

(40) *ChemDraw*; CambridgeSoft: Cambridge, MA.

(41) Chemistry Add-in for Word: Microsoft Research. http://research.microsoft.com/chem4word/ (accessed January 12, 2011).

(42) Corbett, P. Oscar3. http://sourceforge.net/projects/oscar3-chem/ (accessed January 12, 2011).