Chemical Rules and Term Rewriting

Oana Andrei

▶ To cite this version:

HAL Id: inria-00000988
https://hal.inria.fr/inria-00000988

Submitted on 10 Jan 2006

Chemical Rules and Term Rewriting

(Internship Report)

Oana ANDREI

Advisors: Hélène KIRCHNER
Liliana IBĂNESCU
Olivier BOURNEZ

April - September, 2005

Contents

Chapter 1

Introduction

In [BIK05], rule-based programming is explored in the field of automated generation of chemical reaction mechanisms. The authors identify a particular class of graphs as appropriate for the representation of molecules, and they define a graph rewriting relation based on preserving the vertices and changing the edges. By representing cyclic graphs as labelled trees or forests, this particular graph rewrite relation can be simulated by a tree rewriting relation, which can be in turn simulated by a rewriting relation on equivalence classes of terms. Therefore this kind of graph rewriting can be implemented by means of term rewriting. GasEl, a prototype based on this approach, is described in [Iban04], and it is implemented in ELAN making extensive use of the conditional rewriting rules and the built-in strategy language.

The Protheo group also develops since 2002 a system called TOM([TD]), a Pattern Matching Preprocessor that aims at integrating term rewriting and pattern matching facilities into imperative languages such as C and Java.

The subject of the internship consisted in exploring and evaluating the capabilities of TOM to model the graph rewriting relation and to compare this approach with the previous one in ELAN. Also, an expected result was to identify some operations on molecular graphs that can be better implemented using TOM.

The report of the internship begins with a description of the oxidizing pyrolysis as an artificial chemistry: molecules, reactions, and reactor algorithm. Then we prove that the reactor algorithm terminates for a finite number of input chemical species.

We present the implementation of the oxidizing pyrolysis process in TOM: the signature of the chemicals, the reaction rules, and the reactor dynamics. We give two implementations of the reactor dynamics: one for the GasEl-like (or qualitative) approach, and the other for the multiset (or quantitative) approach, also providing a comparison between these two approaches.

In ELAN a named rewriting rule can be applied only at the top of a term. Therefore in GasEl for each reactant `AllVisions` is computed: starting from the associated molecular graph, a molecular tree is obtained by choosing a set of edges to cut, and then a *vision* is obtained by choosing a root for the tree. While using TOM we can perform term traversal by means of traversal strategies, and change a term not only at its top. This result shows that using TOM, the problem of matching a subgraph against a molecular graph and the transformation of the graph is solved in a more elegant and better way than in GasEl.

In GasEl the term encoding a free radical is always considered to have the electron as child of the atom in the root. Whereas we always put the electron directly in the root. The implementation of the reactor dynamics in GasEl uses strategies, while in the current implementation we use Java with some

TOMconstructs and strategies.

In the appendices we present a small user's guide of the implementation, and the results of generating mechanisms for two chemical species using the current implementation.

The report is organized as follows.

Chapter 2 starts with a general definition of the artificial chemistry, then it present the oxidizing pyrolysis as an artificial chemistry: molecules, reactions, and reactor algorithm. We prove that the reactor algorithm terminates for a finite number of input chemical species.

Chapter 3 reviews the notion of *molecular graph*, the model used in automated generation of detailed kinetic mechanisms, and its corresponding term representation.

In Chapter 4 we survey some features of the strategy language provided by ELAN and the way this is used in the GasEl prototype.

Chapter 5 offers a quick insight of TOM: some language constructs, Vas and ApiGen, the mutraveler library for traversal strategies. All these features are illustrated by simple yet consistent examples.

In Chapter 6 we present the implementation of the oxidizing pyrolysis process in TOM: the signature of the chemicals, the reaction rules, and the reactor dynamics. We implement the reactor dynamics following the GasEl approach as well as the multiset approach providing as well a comparison between these two aproaches. This chapter ends with a comparison between the TOM implementation and GasEl prototype.

Chapter 7 draws some conclusions, while Appendix A presents a small user's guide of the implementation, and Appendix B presents the results of generating mechanisms for two chemical species using the current implementation.

This 5 months internship took place within the INRIA International Internship Program framework, from April, 15, until September, 15, 2005, in the PROTHEO team in Nancy, under the supervision of Hélène Kirchner, Liliana Ibanescu, and Olivier Bournez.

Chapter 2

The Oxidizing Pyrolysis as an Artificial Chemistry

2.1   An Artificial Chemistry Definition

The most general definition for artificial chemistry [DZB01] states that:

An artificial chemistry is a man-made system which is similar to a real chemical system.

Formally, an *artificial chemistry* can be defined by a triple $AC = (S, R, A)$ where:

- S is the set of all possible molecules,

- R is a set of collision (or reaction) rules representing the interaction among the molecules,

- A is an algorithm describing the reaction vessel $P \subseteq S$ and how rules are applied to the molecules inside the vessel.

2.2   The Oxidizing Pyrolysis

In this section we describe the oxidizing pyrolysis as an artificial chemistry, by presenting each component in turn.

2.2.1   The Set of Molecules

The set of molecules $S = \{s_1, s_2, \ldots, s_n \ldots\}$ might be infinite, and it describes all valid molecules which may appear in an AC.

The molecules for the current model are called hydrocarbons and they can be classified in four distinct chemical families: alcans, cyclans, alkylaromatics, and alkylcycloaromatics.

The model for representing a molecule is the *molecular graph* [DU73]. We present in detail this model in chapter 3.

2.2.2   The SMILES Notation for Molecules

SMILES (Simplified Molecular Input Line Entry System) [US] is a chemical notation that provides a two-dimensional chemical structure to computer programs. The basic syntax rules used in our approach are detailed in the following:

1. Atoms and Bonds

SMILES supports all elements in the periodic table. An atom is represented using its respective atomic symbol. Upper case letters refer to non-aromatic atoms; lower case letters refer to aromatic atoms. If the atomic symbol has more than one letter, the second letter must be lower case.

Bonds are denoted as follows:

- single bond: **–**

- double bond: **=**

- triple bond: **#**

- aromatic bond: *****

Single bonds are the default, therefore they need not be entered, and the bond between two lower case atom symbols is aromatic by default.

2. Simple Chains

Simple chain structures are represented by combining atomic symbols and bond symbols. The SMILES structures are hydrogen-suppressed, and, if enough bonds are not identified, then the other connections are assumed to be satisfied by hydrogen bonds.

Example:

CC	CH_3CH_3	Ethane
$C = C$	CH_2CH_2	Ethene
CBr	CH_3Br	Bromomethane

However, if one chooses to explicitly identify the hydrogen atoms, then all hydrogen atoms must be identified.

3. Branches

A branch from a chain is specified by placing the SMILES symbols for the branch between parenthesis. The string in parenthesis is placed directly after the symbol for the atom to which it is connected. If it is connected by a double or triple bond, the bond symbol immediately follows the left parenthesis.

Example:

$CC(O)C$	2-Propanol
$CC(= O)C$	2-Propanone
$CC(CC)C$	2-Methylbutane
$CC(C)(C)CC$	2,2-Dimethylbutane

4. Cycles

Cyclic structures are represented by using numbers to identify the opening and the closing cycle atom. For example, in $C1CCCCC1$ the first carbon has a label '1' which connects by a single bond with the last carbon which also has a label '1'. The resulted structure is cyclohexane. Chemicals that have multiple cycles may be identified by using different numbers for each ring. If a double, single, or aromatic bond is used for the ring closure, the bond symbol is placed before the ring closure number.

However, the simple bond symbol is implicit, hence it is not compulsory to appear before the ring closure number. The same convention is available for aromatic bonds because they are encoded by means of lower case atom symbols.

Example:

|	
c1ccccc1	Benzene
C1OC1CC	Ethyloxirane
c1cc2ccccc2cc1	Naphthalene

Unique SMILES

For a given chemical structure, arbitrary SMILES notation can take many equally valid forms. One "unique" notation must emerge among the others to serve as the identifier of the structure. In [WWW89] an algorithm that generates the canonical SMILES notations for arbitrary SMILES notations is presented.

In our current implementation we use the binary sources for this algorithm in order to compare two term represented chemical structures.

2.2.3   The Reactions

The set of reaction rules R describes the interactions between the molecules from S. Generally, a rule $r \in R$ can be written in the form:

$$s_1 + s_2 + \ldots + s_n \rightarrow s'_1 + s'_2 + \ldots + s'_k,$$
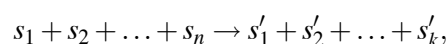
(where instead of "+" we could have used any other symbol with the purpose of separating molecules).

In particular, for the oxidizing pyrolysis n and m are equal to 1 or 2.

R is partitioned in three sets R_{init}, R_{propag}, and R_{termin} where:

- $R_{init} = \{(ui), (bi)\}$

- $R_{propag} = \{(me), (ipso), (bs), (ox), (co.O.)\}$

- $R_{termin} = \{(co), (di)\}$

Figure 2.1 gives the generic elementary reactions (reaction patterns).

Analyzing the left hand side of each generic elementary reaction, we can identify five specific patterns:

- the free electron, denoted by •, in reactions 3 to 9;

- a **C**—**C** bond, in reactions ui (1), ipso (3), and bs (5);

- a **C**—**H** bond, in reactions ui (1), bi (2), me (4), bs (5), ox (6), and di (9);

- an oxygen molecule, **O=O**, in reactions bi (2), and ox (6);

- the biradical •**O**• in reaction co.O. (7).

We continue with a more detailed presentation of the reaction rules illustrating them by appropriate reactions.

	Name	Description
1	unimolecular initiation (ui)	$x - y \longrightarrow x\bullet + \bullet y$
2	bimolecular initiation (bi)	$\mathbf{O} = \mathbf{O} + \mathbf{H} - x \longrightarrow \bullet\mathbf{OOH} + \bullet x$
3	ipso	$\bullet\mathbf{H} + Ar - x \longrightarrow \mathbf{H} - Ar + \bullet x$
4	metathesis (me)	$\bullet\beta + \mathbf{H} - x \longrightarrow \beta - \mathbf{H} + \bullet x$
5	unimolecular decomposition of free radicals by beta-scission (bs)	$\bullet x - y - z \longrightarrow x = y + \bullet z$
6	oxidation of free radicals (ox)	$\mathbf{O} = \mathbf{O} + \mathbf{H} - x - y\bullet \longrightarrow \bullet\mathbf{OOH} + x = y$
7	combination with $\bullet\mathbf{O}\bullet$ (co.O.)	$\bullet\mathbf{O}\bullet + \bullet x \longrightarrow \bullet\mathbf{O} - x$
8	combination of free radicals (co)	$\bullet x + \bullet y \longrightarrow x - y$
9	disproportionation of free radicals (di)	$\bullet x + \mathbf{H} - y - z\bullet \longrightarrow x - \mathbf{H} + y = z$

Figure 2.1: Reaction patterns of primary mechanism given by emphasizing patterns like a simple $(-)$ or double $(=)$ bond, a free radical $(\bullet x)$, a specific atom $(\mathbf{O}, \mathbf{H})$. Symbols different from atom symbols $(\mathbf{C}, \mathbf{O}, \mathbf{H})$ are variables and can be instantiated by any radical

Unimolecular initiation (ui)

A molecule gives rise to two free radicals by breaking a single bond, $\mathbf{C} - \mathbf{C}$ or $\mathbf{C} - \mathbf{H}$:

$$x - y \longrightarrow x\bullet + \bullet y \tag{2.1}$$

Example 2.1 *Unimolecular initiation of ethylbenzene by breaking a $\mathbf{C} - \mathbf{C}$ bond on the alkyl branch produces either two radicals, benzyl and methyl:*

$$\tag{2.2}$$
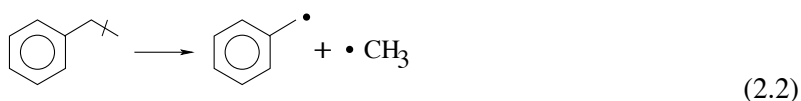
or the radicals phenyl and ethyl:

$$\tag{2.3}$$
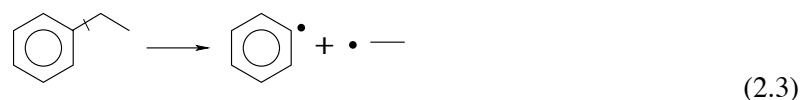
Bimolecular initiation (bi)

An oxygen molecule abstracts a hydrogen atom from an alkylic or cyclo-alkylic substructure x:

$$\mathbf{O} = \mathbf{O} + \mathbf{H} - x \longrightarrow \bullet\mathbf{OOH} + \bullet x \tag{2.4}$$

Example 2.2 *One of the bimolecular initiation for ethylbenzene is the following:*

$$\tag{2.5}$$

8

(ipso)

A free radical $\bullet H$ adds on an aromatic cycle *Ar* and breaks a $\mathbf{C-C}$ bond from an alkylic or cyclo-alkylic branch:

$$\bullet\mathbf{H} + Ar - x \longrightarrow \mathbf{H} - Ar + \bullet x \tag{2.6}$$

Example 2.3 *Ipso reaction for ethylbenzene produces benzene and ethyl:*

$$\bullet H + \text{[structure]} \longrightarrow \text{[structure]} + \bullet - \tag{2.7}$$
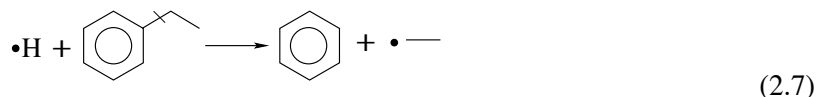
Example 2.4 *Each one of the two ipso reactions for ehyltetraline breaks a bond on the cyclane and produces one radical:*

$$\bullet H + \text{[structure]} \longrightarrow \text{[structure]} \tag{2.8}$$

$$\bullet H + \text{[structure]} \longrightarrow \text{[structure]} \tag{2.9}$$
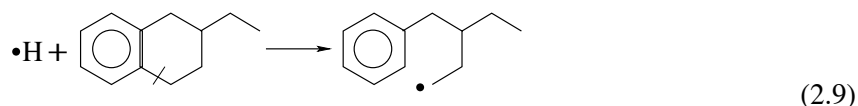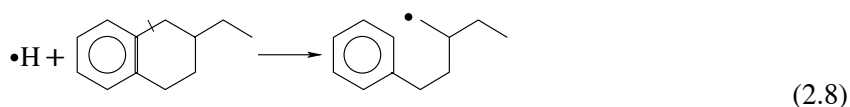
Metathesis (me)

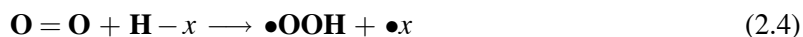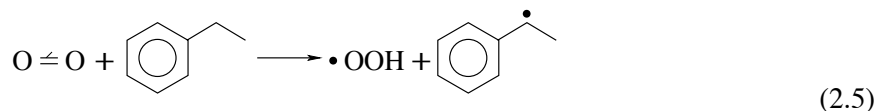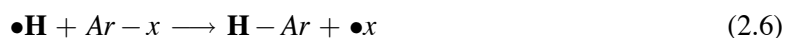A free radical (a β radical) abstracts a hydrogen atom from an alkylic or cyclo-alkylic substructure *x*:

$$\bullet\beta + \mathbf{H} - x \longrightarrow \beta - \mathbf{H} + \bullet x \tag{2.10}$$

Example 2.5 *A metathesis of ethylbenzene with methyl produces methane and a free radical:*

$$\bullet CH_3 + \text{[structure]} \longrightarrow CH_4 + \text{[structure]} \tag{2.11}$$
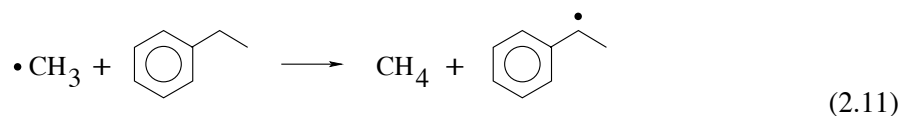
We consider only the following β radicals: $\bullet H$, $\bullet O\bullet$, $\bullet OH$, $\bullet OOH$, $\bullet CH_3$.

Unimolecular decomposition of free radicals by beta-scission (bs)

This reaction occurs by breaking a single bond in the β position (at distance two) with respect to the radical point with the concomitant formation of a double bond.

$$\bullet x - y - z \longrightarrow x = y + \bullet z \tag{2.12}$$

Example 2.6 *Unimolecular decomposition by beta-scission by breaking a $\mathbf{C-C}$:*

$$\text{[structure]} \longrightarrow \text{[structure]} + \bullet CH_3 \tag{2.13}$$

9
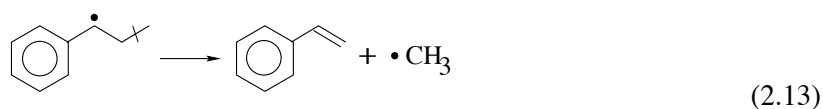
and by breaking a **C−H** *bond:*

$$\text{(2.14)}$$
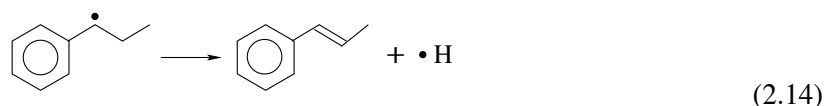
Oxidation of free radicals (ox)

An oxygen molecule abstracts a hydrogen atom situated in the β position (at distance two) with respect to the radical point:

$$\mathbf{O} = \mathbf{O} + \mathbf{H} - x - y\bullet \longrightarrow \bullet\mathbf{OOH} + x = y \tag{2.15}$$

Example 2.7

$$\text{(2.16)}$$
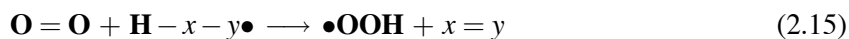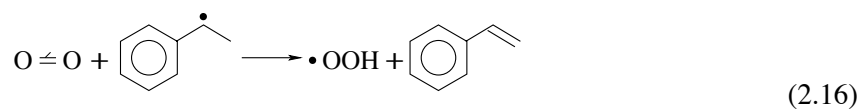
Combination with •O• (co.O.)

The combination of a free radical with the biradical •O• give rise to a new radical:

$$\bullet\mathbf{O}\bullet + \bullet x \longrightarrow \bullet\mathbf{O} - x \tag{2.17}$$

Example 2.8 *The combination of benzyl with the biradical* •O•:

$$\text{(2.18)}$$
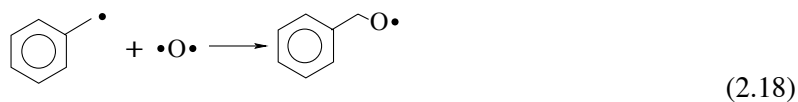
Combination of free radicals (co)

Two free radicals give rise to a molecule by the formation of a covalent bond:

$$\bullet x + \bullet y \longrightarrow x - y \tag{2.19}$$

Example 2.9 *The combination of the benzyl radical with the* •**OOH** *radical produces a hydroperoxyde:*

$$\text{(2.20)}$$
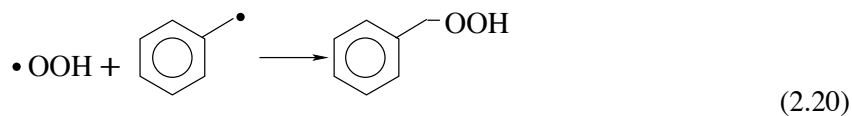
Disproportionation of free radicals (di)

A free radical •*x* abstracts a hydrogen atom situated in β position with respect to the radical point in another free radical:

$$\bullet x + \mathbf{H} - y - z\bullet \longrightarrow x - \mathbf{H} + y = z \tag{2.21}$$
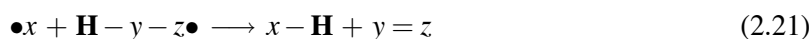
		Reaction	C—H C∉Ar	C—C ∉cycle	C—C ∉Ar
1	ui	$x-y \longrightarrow x\bullet + \bullet y$	*	*	*
2	bi	$\mathbf{O=O} + \mathbf{H}-x \longrightarrow \bullet\mathbf{OOH} + \bullet x$	*		
3	ipso	$\bullet\mathbf{H} + Ar-x \longrightarrow \mathbf{H}-Ar + \bullet x$			*
4	me	$\bullet\beta + \mathbf{H}-x \longrightarrow \beta-\mathbf{H} + \bullet x$	*		
5	bs	$\bullet x-y-z \longrightarrow x=y + \bullet z$	*		*
6	ox	$\mathbf{O=O} + \mathbf{H}-x-y\bullet \longrightarrow \bullet\mathbf{OOH} + x=y$	*		
7	co.O.	$\bullet\mathbf{O}\bullet + \bullet x \longrightarrow \bullet\mathbf{O}-x$			
8	co	$\bullet\beta + \bullet\mathrm{Y} \longrightarrow \beta-\mathrm{Y}$ $\bullet\mathrm{Y} + \bullet\mathrm{Y}' \longrightarrow \mathrm{Y}-\mathrm{Y}'$			
9	di	$\bullet\beta + \mathbf{H}-y-z\bullet \longrightarrow \beta-\mathbf{H} + y=z$	*		

Figure 2.2: Filters for generic elementary reactions

Example 2.10 *Disproportionation of the benzyl radical with the •OOH radical produces ethylbenzene and oxygen:*

$$\bullet \mathrm{OOH} + \quad \longrightarrow \quad + \mathrm{O=O} \tag{2.22}$$
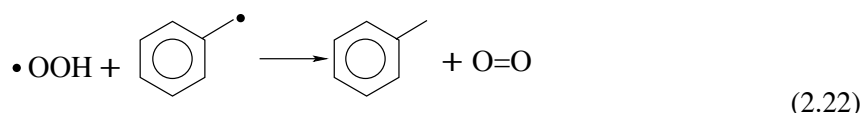
Chemical Filters

For the reaction patterns already given the following restrictions should be imposed:

- there is no **C—H** bond breaking if the **C** atom belongs to an aromatic cycle.

- there is no **C—C** bond breaking if this bond is on an aromatic cycle.

- for (ui) there is no **C—C** bond breaking if this bond is on a cycle.

The restrictions for all rules are summarized in Figure 2.2 also considering the following two remarks. In the first place, the free radicals belonging to the classes of beta radicals and Y radicals respectively are directly encoded in the reaction rules. In the second place, for (co.O.) there are some specific restrictions:

- the atom labelled by x cannot be **O**;

- the number of oxygen atoms in the molecular graph which contains the atom labelled by x is limited (to two, for example).

2.2.4 The Reactor Algorithm

A *reactor algorithm A* determines how a set of reactions rules R is applied to a collection (usually a multiset) P of molecules from S. The output consists of the lists of elementary reactions applied and/or the list of final reaction products.

In the following we present a general approach of how the dynamics of a reaction vessel can be modelled and simulated. The algorithm A depends on the representation of P. In this approach every molecule is explicitly simulated and the population is represented as a multiset P. A typical algorithm

draws a sample of molecules randomly from the population P and checks whether a rule $r \in R$ can be applied. If so, the molecules are replaced by the right hand side molecules given by r. If more than one rule can apply, a decision is implemented which rule to employ. If no rule can be applied, a new random drawing is initialized. The algorithm is not necessarily restricted to be such simple. Further parameters such as rate constants, energy, spatial information, or temperature can be introduced into the rules for the chemistry to become more realistic.

The following example is a general reactor algorithm used for an AC with second-order reactions only:

Algorithm 2.1 The reactor dynamics:

$$
\begin{aligned}
&\textbf{while}(\neg terminate()) \textbf{ do} \\
&\quad m_1 := draw(P); \\
&\quad m_2 := draw(P); \\
&\quad \textbf{if } \exists (m_1 + m_2 \rightarrow m'_1 + m'_2 + \ldots + m'_k) \in R \\
&\quad\quad \textbf{then} \\
&\quad\quad\quad P := remove(P, m_1, m_2); \\
&\quad\quad\quad P := insert(P, m'_1, m'_2, \ldots, m'_k); \\
&\quad \textbf{fi} \\
&\textbf{od}
\end{aligned}
$$

The function *draw* returns a randomly chosen molecule from P without removing it from P.

The most common method used for simulating a reactional mechanism consist in an iterative method:

Let P_0 be the multiset of input molecules, and $i = 0$.

(1) the molecules from P_i interact according to a set of considered reaction rules R, and the resulted molecules are collected in a new multiset P'_{i+1};

(2) the result of the iteration is $P_{i+1} = P'_{i+1} \setminus P_i$, therefore P_i contains only the newly created molecules;

(3) i is incremented, and we iterate (1) and (2) until the process is stabilized, i.e., until P_i becomes empty.

For the oxidizing pyrolysis the reactor algorithm is called *the primary mechanism* and it consists of three stages:

1. *initiation stage*: unimolecular and bimolecular initiation reactions are applied to the initial reactants. This phase produces free radicals.

2. *propagation stage*: a set of generic patterns of reactions are applied to free radicals producing new free radicals. This propagation phase is iterated until no new free radical is generated. The generic patterns of reactions implemented in the GasEl prototype are: (ipso), metathesis (me), beta-scission (bs), oxidation (ox), and combination of a radical with the biradical •O• (co.O.).

3. *termination stage*: combination and disproportionation are applied to free radicals to get a set of molecules.

We consider that the three stages of the reactor are executed sequentially due to chemical hypothesis. The reactions can be executed in parallel as well if they satisfy the following conditions:

- (ui) and (bi) can be applied only on input molecules;

- (co) and (di) can be applied only when no propagation rule can be applied anymore.

For expository reasons we consider that all reactions have the generic form $m_1 + m_2 \rightarrow m'_1 + m'_2$, where at most one reactant in each side of the rule can be a "dummy" reactant which is always present in a chemical soup.

Algorithm 2.2 *[Initiation]* The reactor dynamics during the initiation stage:
    Input: P_0, $P_1 := P_0$
    Output: P_1

$$
\begin{aligned}
&\textbf{while}(\neg terminate()) \textbf{ do} \\
&\quad (m_1, m_2) := select(P_0); \\
&\quad\quad \textbf{for all } (m_1 + m_2 \rightarrow m'_1 + m'_2) \in R_{init} \\
&\quad\quad\quad P_1 := insert(P_1, m'_1, m'_2); \\
&\quad\quad \textbf{fi} \\
&\textbf{od}
\end{aligned}
$$

The function $select(P)$ returns a randomly chosen pair molecule from P not chosen before.

We chose here to present the GasEl approach, i.e., we do not remove the molecules that react. However, in the multiset approach, each molecule has an associated number of occurrences in the soup or population, and the reactions *consume* the reactants, i.e., the multiplicity of reactants is decreased each time. We can consider that the GasEl approach is an abstraction of the multiset approach by assuming that there are *enough* quantities of each species of molecules such that the molecules interact as much as possible.

In GasEl *small molecules* (i.e., molecules that contain at most one carbon atom, like $\mathbf{O} = \mathbf{O}$, $\bullet\mathbf{H}$, β radicals, $\bullet\mathbf{O}\bullet$, etc.) can always be injected in the chemical soup or we assume they exist in order to allow reactions like (bi), (me), (ipso), (ox), (co.O.) to be applied. On the contrary in our implementation we always work with an explicit population of reactants.

The function $insert(P, m'_1, m'_2)$ inserts the two reaction products m'_1 and m'_2 in P according to its structure. If the element to be inserted is already in P, then in the set case P does not change, while in the multiset case the multiplicity of the element is increased or the element is added to P. If the element is not already in P, it is simply added (with the multiplicity 0 in the multiset case).

For describing the propagation stage we use an iterative method. Let P_0 be the multiset of input molecules, $i := 0$, and $P'' := P_0$.

(1) the molecules from P_i interact according to a set of considered reaction rules R, and the resulted molecules are collected in a new multiset P'_{i+1};

(2) the result of the iteration is $P_{i+1} = P'_{i+1} \setminus P''$, and it is added to P''; P_{i+1} contains only the newly created molecules which are going to become subjects for propagation rules, hence we can call each P_i an *active* soup of reactants;

(3) i is incremented, and we iterate (1) and (2) until the process is stabilized, i.e., until P_{i+1} becomes empty.

Algorithm 2.3 *[Propagation]* The reactor dynamics during the propagation stage:
    Input: $P_0 := P_1$, $i := 0$, $P'' := \emptyset$
    Output: P''

```
repeat
    P'' := P'' ∪ P_i;
    while(¬terminate()) do
        (m_1, m_2) := select(P_i);
        for all (m_1 + m_2 → m'_1 + m'_2) ∈ R_propag
            P_{i+1} := insert(P_{i+1}, m'_1, m'_2);
        fi
    od
    P_{i+1} := P_{i+1} \ P'';
    i := i + 1;
until P_i = ∅
```

The algorithm for the termination stage is almost identical with the one for initiation, except that instead of initiation rules we consider termination rules.

Algorithm 2.4 *[Termination]* The reactor dynamics during the termination stage:

Input: $P_0 = P''$, $P_1 = \emptyset$
Output: P_1

```
while(¬terminate()) do
    (m_1, m_2) := select(P_0);
    for all (m_1 + m_2 → m'_1 + m'_2) ∈ R_termin
        P_1 := insert(P_1, m'_1, m'_2);
    fi
od
```

We can notice that all three algorithms have a common part parameterized after a set of reaction rules and an input chemical soup:

```
P(S) UNIT (R_1 : P(R), P_1 : P(S))
    begin
        P_2 := ∅;
        while(¬terminate()) do
            (m_1, m_2) := select(P_1);
            for all (m_1 + m_2 → m'_1 + m'_2) ∈ R_1
                P_2 := insert(P_2, m'_1, m'_2);
            fi
        od
        return P_2
    end
```

Now the algorithms for the three stages can be written as follows:

$P(S)$ AlgInit $(P_0 : P(S))$
    begin
        return $P_0 \cup \text{UNIT}(R_{init}, P_0)$
    end

$P(S)$ AlgTermin $(P_0 : P(S))$
    begin
        return $P_0 \cup \text{UNIT}(R_{termin}, P_0)$
    end

$P(S)$ AlgPropag $(P_0 : P(S))$
    begin
        $i := 0;$
        $P'' := \emptyset;$
        repeat
          $P'' := P'' \cup P_i;$
          $P_{i+1} := \text{UNIT}(R_{propag}, P_i) \setminus P'';$
          $i := i + 1;$
        until $P_i = \emptyset;$
        return $P'';$
    end

The Reactor Algorithm Terminates

We show that the algorithm presented above terminates by showing that the algorithms for each stage terminate.

For AlgInit the function $terminate()$ returns $false$ as long as there are molecules that can react by means of reaction rules from R_{init}. It will eventually return $true$ because:

- there is a finite number of molecules in the initial set P_0, and

- each molecule has a finite number of bonds and atoms, therefore the number of reactions (ui) and (bi) applied on a molecule is also finite.

For AlgTermin the function $terminate()$ will eventually return $true$ because:

- there is a finite number of free radicals in the soup,

- the termination rules take two free radicals and produce a molecule; hence the reactions apply only on the existing radicals and not on the reaction products.

In the following we show that AlgPropag terminates.

Analyzing the chemical filters, we can note that the application of (co.O.) on a radical is either followed by an application of (bs) or by an application of (ox), either the resulted radical is irreducible w.r.t. the propagation rules.

A radical $\bullet O - C - y$ resulted from (co.O.) is *irreducible w.r.t. the propagation rules* if it is irreducible w.r.t. (bs), (ox), and (co.O.) respectively. Moreover, according to chemical filters, $\bullet O - C - y$ is:

(bs)-irreducible if either the carbon atom is aromatic or it has no hydrogen bonds, either y has a non carbon label or the carbon atom is aromatic and y has as well an aromatic carbon label;

(ox)-irreducible if the carbon atom is aromatic or it has no hydrogen bonds;

(co.O.)-irreducible because the radical point is attached to an oxygen labelled atom.

This remark allows us to consider two new reaction rules resulting from the compositional of (co.O.) with (bs) and with (ox) respectively. Let ; denote the composition of two reaction rules, and (co.O.;bs) and (co.O.;ox) denote the two new rules.

The composed reaction

$$\bullet O \bullet + \bullet C - y \xrightarrow{(co.O.)} \bullet O - C - y \xrightarrow{(bs)} O = C + \bullet y \tag{2.23}$$
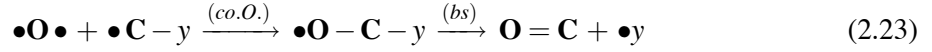
becomes the atomic reaction (co.O.;bs):

$$\bullet O \bullet + \bullet C - y \xrightarrow{(co.O.;bs)} O = C + \bullet y \tag{2.24}$$

while the composed reaction below

$$\bullet O \bullet + \bullet x - H \xrightarrow{(co.O.)} \bullet O - x - H \xrightarrow[+O=O]{(ox)} \bullet OOH + O = x \tag{2.25}$$

becomes the atomic reaction (co.O.;ox):

$$\bullet O \bullet + \bullet x - H \xrightarrow{(co.O.;ox)} \bullet OOH + O = x \tag{2.26}$$
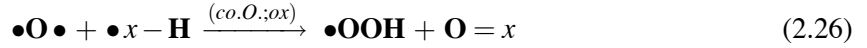
Moreover, we can move the reaction (co.O) in a separate stage between the propagation and the termination stages. Let R' denote the new set of propagation rules:

$$R' = \{(me), (ipso), (bs), (ox), (co.O.;bs), (co.O.;ox)\}$$

Let $k(m)$ denote the number of elementary cycles in the molecular graph corresponding to reactant m. For x and y two atoms, we denote by $x \frown y$ a chain of at least two edges between the nodes corresponding to x and y in the molecular graph, and by $x \simeq y$ both $x - y$ and $x \frown y$. We define *the number of simple bonds* for a soup of reactants as follows:
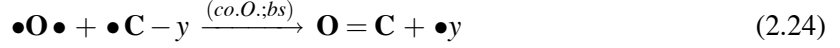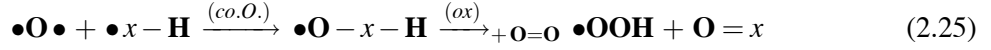
$$
\begin{aligned}
\sigma(m_1 + \ldots + m_k) &= max\{\sigma(m_1), \ldots, \sigma(m_i), \ldots, \sigma(m_k)\} \\
\sigma(m) &= 0, \text{ if } m \text{ is a small molecule} \\
\sigma(m_1 - m_2) &= \sigma(m_1 \bullet) + \sigma(\bullet m_2) + 1 \\
\sigma(m_1 = m_2) &= \sigma(\bullet m_1 \bullet) + \sigma(\bullet m_2 \bullet) \\
\sigma(m_1 \# m_2) &= \sigma(\bullet \dot{m}_1 \bullet) + \sigma(\bullet \dot{m}_2 \bullet) \\
\sigma(m_1 \simeq m_2) &= \sigma(\bullet m_1 \frown m_2 \bullet) + 1
\end{aligned}
$$

Remark 2.1   *1. σ can also be defined for free radicals, but this is not needed for the proof of Proposition 2.1.*

2. Due to chemical hypotheses, two small molecules do not interact.

Proposition 2.1 *If $S_1 \longrightarrow S_2 \in R'$ then:*

1. *$\sigma(S_1) \geq \sigma(S_2)$,*

2. *$k(S_1) \geq k(S_2)$, and*

3. *$(\sigma + k)(S_1) > (\sigma + k)(S_2)$.*

Proof: We prove the three inequalities for each type of rule from R':

16

(me) $\bullet\beta + \mathbf{H} - x \longrightarrow \beta - \mathbf{H} + \bullet x$

$\sigma(S_1) = \sigma(\bullet\beta + \mathbf{H} - x) = max\{\sigma(\bullet\beta), \sigma(\mathbf{H} - x)\} = max\{0, \sigma(\mathbf{H} - x)\} = \sigma(\mathbf{H} - x) = \sigma(\mathbf{H}\bullet) + \sigma(\bullet x) + 1 = \sigma(\bullet x) + 1$

$\sigma(S_2) = \sigma(\beta - \mathbf{H} + \bullet x) = max\{\sigma(\beta - \mathbf{H}), \sigma(\bullet x)\} = max\{0, \sigma(\bullet x)\} = \sigma(\bullet x)$

Therefore $\sigma(S_1) > \sigma(S_2)$.

As cycles in molecular graphs do not contain hydrogen atoms, and this reaction breaks a **C–H** bond, this reaction does not involve cycle breaking. Therefore $k(S_1) = k(S_2)$.

(ipso) $\bullet\mathbf{H} + Ar - x \longrightarrow \mathbf{H} - Ar + \bullet x$

(i) The bond $\mathbf{c} - x$ is not component of a cycle in the associated molecular graph

$\sigma(S_1) = \sigma(\bullet\mathbf{H} + Ar - x) = max\{\sigma(\bullet\mathbf{H}), \sigma(Ar - x)\} = max\{0, \sigma(Ar - x)\} = \sigma(Ar - x) = \sigma(Ar\bullet) + \sigma(\bullet x) + 1$

$\sigma(S_2) = \sigma(\mathbf{H} - Ar + \bullet x) = max\{\sigma(\mathbf{H} - Ar), \sigma(\bullet x)\} = max\{\sigma(\mathbf{H} - Ar), \sigma(\bullet x)\} = max\{\sigma(\mathbf{H}\bullet) + \sigma(\bullet Ar) + 1, \sigma(\bullet x)\} = max\{\sigma(\bullet Ar) + 1, \sigma(\bullet x)\}$

If $\bullet x \in B$ then $\sigma(\bullet x) = 0$, but $\sigma(\bullet x) > 0$ because the radical point is attached to a carbon atom. Therefore $\sigma(S_2) = \sigma(\bullet Ar) + 1 < \sigma(Ar\bullet) + \sigma(\bullet x) + 1 = \sigma(S_1)$.

Otherwise, if $\bullet x \notin B$ then $\sigma(\bullet x) > 0$. Therefore $\sigma(S_2) = max\{\sigma(\bullet Ar) + 1, \sigma(\bullet x)\} < \sigma(\bullet Ar) + 1 + \sigma(\bullet x) = \sigma(S_1)$.

In this case it is obviously that $k(S_1) = k(S_2)$.

(ii) The bond $\mathbf{c} - x$ is component of a cycle in the associated molecular graph. Then we can write the reaction rule as:

$$\bullet\mathbf{H} + Ar \frown x \longrightarrow \mathbf{H} - Ar \frown x\bullet$$

$\sigma(S_1) = \sigma(\bullet\mathbf{H} + Ar \frown x) = max\{\sigma(\bullet\mathbf{H}), \sigma(Ar \frown x)\} = \sigma(Ar \frown x) = \sigma(\bullet Ar \frown x\bullet) + 1$

$\sigma(S_2) = \sigma(\mathbf{H} - Ar \frown x\bullet) = \sigma(\mathbf{H}\bullet) + \sigma(\bullet Ar \frown x\bullet) + 1 = \sigma(\bullet Ar \frown x\bullet) + 1 = \sigma(S_1)$

Therefore on one side we have $\sigma(S_1) = \sigma(S_2)$, but on the other side we have $k(S_1) > k(S_2)$ because by breaking the bond $\mathbf{c} - x$ the cycle is also broken.

(bs) $\bullet x - y - z \longrightarrow x = y + \bullet z$

(i) The nodes corresponding to x and z, and to y and z respectively are not on a cycle in the associated molecular graph.

$\sigma(S_1) = \sigma(\bullet x - y - z) = \sigma(\bullet x - y\bullet) + \sigma(\bullet z) + 1$

$\sigma(S_2) = \sigma(x = y + \bullet z) = max\{\sigma(x = y), \sigma(\bullet z)\} = max\{\sigma(x = y), \sigma(\bullet z)\} \leq \sigma(x = y) + \sigma(\bullet z)$

From $\sigma(x = y) = \sigma(\bullet x\bullet) + \sigma(\bullet y\bullet)$ and $\sigma(\bullet x - y\bullet) = \sigma(\bullet x\bullet) + \sigma(\bullet y\bullet) + 1$, we obtain $\sigma(x = y) < \sigma(\bullet x - y\bullet)$. Therefore $\sigma(S_1) > \sigma(S_2)$.

By breaking the bond $y - z$ not on a cycle, we have $k(S_1) = k(S_2)$.

(ii) The bond $y - z$ is component of a cycle in the associated molecular graph, and the node corresponding to x is not on this cycle. Then we can write the reaction rule as:
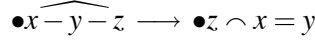
$$\bullet x - y \frown z \longrightarrow x = y \frown z\bullet$$

$\sigma(S_1) = \sigma(\bullet x - y \frown z) = \sigma(\bullet x - y\bullet \frown z\bullet) + 1 = \sigma(\bullet x\bullet) + \sigma(\bullet y\bullet \frown z\bullet) + 2$

$\sigma(S_2) = \sigma(x = y \frown z\bullet) = \sigma(\bullet x\bullet) + \sigma(\bullet y\bullet \frown z\bullet) < \sigma(S_1)$

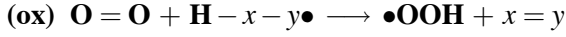A cycle is broken in the associated molecular graph by breaking the bond $y - z$, hence $k(S_1) > k(S_2)$.

(iii) The nodes corresponding to x, y, and z are situated on a cycle in the associated molecular graph. Then the reaction rule has the form:

$$\bullet x \overset{\frown}{- y - z} \longrightarrow \bullet z \frown x = y$$

$\sigma(S_1) = \sigma(\bullet x \overset{\frown}{- y - z}) = \sigma(\bullet z \frown \bullet x - y\bullet) = \sigma(\bullet z \frown \bullet x\bullet) + \sigma(\bullet y\bullet) + 1$

$\sigma(S_2) = \sigma(\bullet z \frown x = y) = \sigma(\bullet z \frown \bullet x\bullet) + \sigma(\bullet y\bullet) < \sigma(S_1)$

In this case there is also a cycle break, therefore $k(S_1) > k(S_2)$.

(ox) $\mathbf{O = O + H} - x - y\bullet \longrightarrow \bullet\mathbf{OOH} + x = y$

$\sigma(S_1) = \sigma(\mathbf{O = O + H} - x - y\bullet) = max\{\sigma(\mathbf{O = O}), \sigma(\mathbf{H} - x - y\bullet)\} = \sigma(\mathbf{H} - x - y\bullet) = \sigma(\mathbf{H} - x - y\bullet) = \sigma(H\bullet) + \sigma(\bullet x - y\bullet) + 1 = \sigma(\bullet x - y\bullet) + 1 = \sigma(\bullet x\bullet) + \sigma(\bullet y\bullet) + 2$

$\sigma(S_2) = \sigma(\bullet\mathbf{OOH} + x = y) = max\{\sigma(\bullet\mathbf{OOH}), \sigma(x = y)\} = \sigma(x = y) = \sigma(x = y) = \sigma(\bullet x\bullet) + \sigma(\bullet y\bullet) < \sigma(S_1)$

This reaction breaks a hydrogen bond which is not on a cycle. Therefore $k(S_1) = k(S_2)$.

(co.O.;bs) $\bullet\mathbf{O}\bullet + \bullet\mathbf{C} - y \rightarrow \mathbf{O = C} + \bullet y$

(i) The bond $\mathbf{C} - y$ is not on a cycle in the associated molecular graph.

$\sigma(S_1) = \sigma(\bullet\mathbf{O}\bullet + \bullet\mathbf{C} - y) = max\{\sigma(\bullet\mathbf{O}\bullet), \sigma(\bullet\mathbf{C} - y)\} = \sigma(\bullet\mathbf{C} - y) = 3 + \sigma(\bullet y)$

$\sigma(S_2) = \sigma(\mathbf{O = C} + \bullet y) = max\{\sigma(\mathbf{O = C}), \sigma(\bullet y)\} = max\{2, \sigma(\bullet y)\} \leq 2 + \sigma(\bullet y) \leq 2 + \sigma(\bullet y) < \sigma(S_1)$
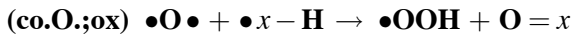
The bond breaking does not involves a cycle breaking, hence $k(S_1) = k(S_2)$.

(ii) The bond $\mathbf{C} - y$ is on a cycle in the associated molecular graph. Then the reaction has the form: $\bullet\mathbf{O}\bullet + \bullet\mathbf{C} \cong y \rightarrow \mathbf{O = C} \frown y\bullet$

$\sigma(S_1) = \sigma(\bullet\mathbf{O}\bullet + \bullet\mathbf{C} \cong y) = max\{\sigma(\bullet\mathbf{O}\bullet), \sigma(\mathbf{C} \cong y)\} = \sigma(\mathbf{C} \cong y) = \sigma(\bullet\mathbf{C} \frown y\bullet) + 1$

$\sigma(S_2) = \sigma(\mathbf{O = C} \frown y\bullet) = \sigma(\bullet\mathbf{O}\bullet) + \sigma(\bullet\dot{\mathbf{C}} \frown y\bullet) = \sigma(\bullet\dot{\mathbf{C}} \frown y\bullet) < \sigma(\bullet\mathbf{C} \frown y\bullet) < \sigma(S_1)$

A cycle is broken during this reaction, hence $k(S_1) > k(S_2)$.

(co.O.;ox) $\bullet\mathbf{O}\bullet + \bullet x - \mathbf{H} \rightarrow \bullet\mathbf{OOH} + \mathbf{O} = x$

$\sigma(S_1) = \sigma(\bullet\mathbf{O}\bullet + \bullet x - \mathbf{H}) = max\{\sigma(\bullet\mathbf{O}\bullet), \sigma(\bullet x - \mathbf{H})\} = \sigma(\bullet x - \mathbf{H}) = \sigma(\bullet x\bullet) + 1$

$\sigma(S_2) = \sigma(\bullet\mathbf{OOH} + \mathbf{O} = x) = max\{\sigma(\bullet\mathbf{OOH}), \sigma(\mathbf{O} = x)\} = \sigma(\mathbf{O} = x) = \sigma(\bullet\mathbf{O}\bullet) + \sigma(\bullet x\bullet) = \sigma(\bullet x\bullet) < \sigma(S_1)$

A hydrogen bond is broken during this reaction, hence no cycle is broken: $k(S_1) = k(S_2)$.

$\square$

We say that a reactant is *propag-irreducible* if it cannot be the subject of any propagation rule.

During the propagation stage we can associate to each (signifiant) reactant a binary evolution tree which has as root the reactant itself, while the rest of the nodes correspond to the reaction products.

Proposition 2.2 *During the propagation stage, every evolution tree is finite.*

Proof: From the propagation algorithm we saw that when we obtain a reactant already in the chemical soup we do not apply reaction rules on it.

For each chain starting from the root in an evolution tree we can associate a strictly descending chain of natural numbers corresponding to the values $(\sigma + k)$ computes for the reactants in the nodes. Because the set of natural numbers is well-formed, this chain is finite and its last element corresponds to a propag-irreducible or an already existing or previously obtained product.

$\square$

This result can be easily extended for the set of propagation rules $R' = R' \cup \{(bs2), (ox2)\}$ where (bs2) and (ox2) are the following two reaction rules:

$$(bs2) \quad \bullet x = y - z \longrightarrow x\#y + \bullet z$$

$$(ox2) \; \mathbf{O} = \mathbf{O} + \mathbf{H} - x = y\bullet \longrightarrow \bullet\mathbf{OOH} + x\#y$$
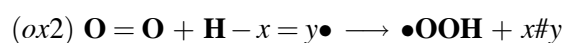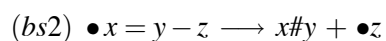
Chapter 3

Chemical Graph Rewriting and Term Rewriting

The model used in automated generation of detailed kinetic mechanisms is the *molecular graph* [DU73], a vertex-labelled and edge-labelled graph, where each vertex is labelled with an atom and each edge is labelled with the bond type, as given in Figure 3.1.
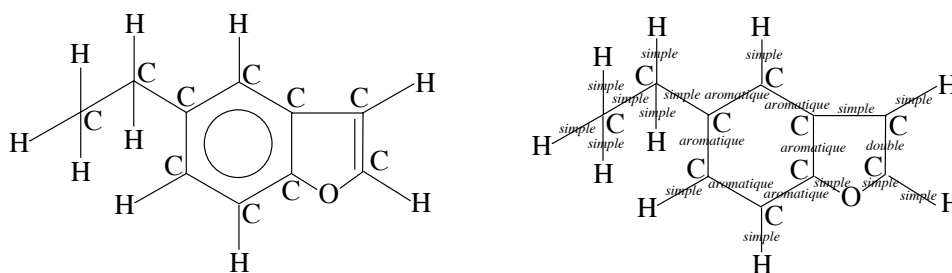
Figure 3.1: A molecular graph

A molecular graph (see Figure 3.1) is encoded by a term, as proposed in the linear notation SMILES presented in [WWW89]. We briefly recall the principles of this representation:

1. Molecules are represented as hydrogen-suppressed molecular graphs (hydrogen atoms are not represented).

2. If the hydrogen-suppressed molecular graph has cycles, we transform it into a tree applying the following rule to every cycle: choose one fresh digit and one single or aromatic bond of the cycle, break the bond and label the 2 atoms with the same digit.

3. Choose a root of the tree, and represent it like a concatenation of the root and the list of its sons.

Some conventions: an aromatic cycle is represented by lower case letters, and a simple bond is not represented.

The formal definitions for encoding a molecular graph by a GasEl term are given in [BIK05]; here we give an example. In the molecular graph from Figure 3.1 two edges are transformed into implicit edges: (i) edge {6,11} labelled with *simple* is hidden and the representation from Figure 3.2 is obtained; (ii) edge {5,6} labelled with *aromatic* is hidden and the result is given in Figure 3.3.
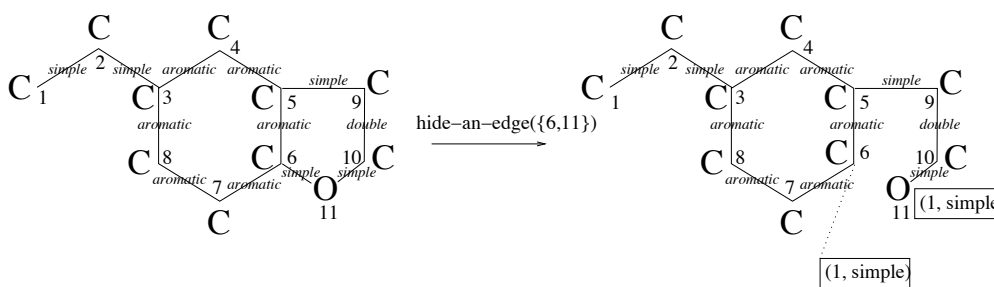
Figure 3.2: Hiding edge {6,11} and encoding it by labels (1, simple) on vertices 6 and 11
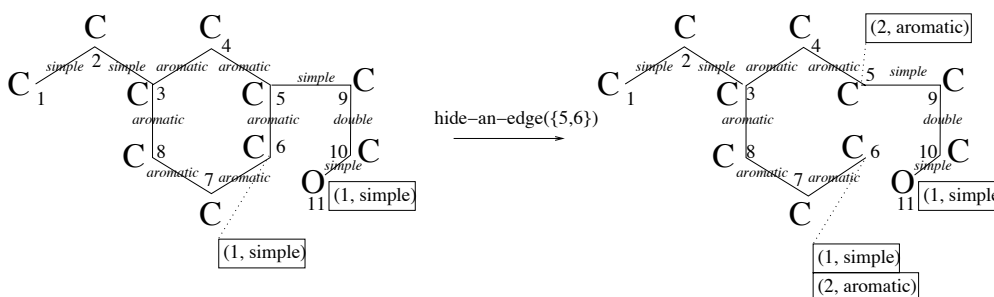
Figure 3.3: Hiding edge {5,6} and encoding it by labels (2, aromatic) on vertices 5 and 6

In Figure 3.3, if the root is chosen as the vertex number 1, then a possible linear notation is `CCc(ccc12)cc2C=CO1`; if the root is the vertex number 3 then an other valid notation is `C(CC)(ccc12)cc2C=CO1`.

If chemical species are represented as molecular graphs, then coding a generic reaction means

1. to describe the generic reaction (the transformation for a class of molecules) as a pattern on molecular graphs,

2. to express conditions on application/non-applications (chemical filters) of this generic reaction, and

3. to apply the generic reaction to specific molecular graphs.

As the molecular graphs are encoded as tree-like structures, the generic reactions are encoded as tree transformations [BIK05].

Chapter 4

The GasEl System

The GasEl system is based on ELAN system [BKK+98], and it generates kinetic mechanisms of fuel combustion. This is one of the objectives of a research project that involved two teams from Nancy, France: one team of computer scientists from LORIA[1] and a team of chemists from DCPR[2] that developed the kinetic mechanism generator system EXGAS [Côm01, WBLF+00].

The ELAN system provides an environment for specifying and prototyping deduction systems in a language based on rules controlled by strategies. ELAN takes from functional programming the concept of abstract data types and the function evaluation principle based on rewriting. But rewriting is inherently non-deterministic since several rules can be applied at different positions in a term, and a computation can have several results. One of the main originality of the language is to provide strategy constructors to specify whether a function call returns several, at least one or only one result. A labelled rule is the most elementary strategy and it is called a primal strategy. The strategy language offers primitives for sequential composition, iteration, deterministic and non-deterministic choices of elementary strategies that are labelled rules:

- the concatenation operator denoted ; builds the sequential composition of two strategies S_1 and S_2. The strategy $S_1; S_2$ fails if S_1 fails, otherwise it returns all results of S_2 applied to the results of S_1.

- the dk operator (*don't know choose*) with a variable arity takes all strategies given as arguments, and it returns for each of them the set of all its results. $dk(S_1, \ldots, S_n)$ fails if all strategies $S_1, \ldots, S_n$ fail.

- the dc operator (*don't care choose*) with a variable arity selects only one strategy that does not fail among its arguments and it returns all its results. $dc(S_1, \ldots, S_n)$ fails if all strategies $S_1, \ldots, S_n$ fail.

- the first operator selects the first strategy that does not fail among its arguments, and it returns all its results. $first(S_1, \ldots, S_n)$ fails if all strategies $S_1, \ldots, S_n$ fail.

- id is the identity strategy that does nothing, and never fails.

- fail always fails and returns an empty set of results.

---

[1] LORIA is the Lorraine Laboratory for Research into Information Technology and its Applications
[2] DCPR is the Department of Physical Chemistry of Reactions

- repeat*(*S*) iterates the strategy *S* until it fails and then returns the last obtained result. repeat*(*S*) never fails and terminates only when *S* fails.

- iterate*(*S*) is similar to repeat*(*S*), except that it returns all intermediate results of succesive applications of *S*.

ELAN has some good properties for the generation of kinetic mechanisms: chemical reactions are naturally expressed using conditional rules, themselves easily understood by chemists; ELAN matching power allows for retrieving patterns in chemical species, thanks to the capability of handling multiset structures through the use of associative and commutative functions; The strategy constructors to define control on rules appears as essential for designing generation mechanisms in a flexible way. Moreover, thanks to its efficient compiler, ELAN can handle a large number of rules application (several thousands per second) and is thus well-suited to the computational complexity of the modelling.

Of course, some technical difficulties arose. One of them is that cyclic molecules are easily represented by graphs whereas ELAN can only do term rewriting. Another one is elimination of redundancies that requires intelligent search in huge data sets.

The GasEl module is made up of three sections:

- a set of basic reactions,

- a generator of detailed primary mechanisms, and

- a generator of lumped secondary mechanisms.

GasEl system accepts as input molecules in SMILES notation; the adequate signature is defined in [BCC+03].

In GasEl generic reactions are encoded by a set of conditional rewriting rules on GasEl terms. The description of each generic pattern of reactions (generic elementary reactions) is given in section 2.2.3.

Using the power of strategies of ELAN, the primary mechanism is defined in a natural way. This corresponds to the concatenation of three strategies corresponding to each phase, `tryInit` for the initiation phase, `tryPropag` for the propagation phase and `tryTermin` for the termination phase:

```
[] mec_prim => tryInit; tryPropag; tryTermin end
```

The user defined strategies `tryInit` and `tryTermin` are easily expressed using the ELAN choice strategy operator **dk** applied to the strategies (the ELAN rewrite rule) defining the generic reactions.

```
[] tryInit   =>  dk(ui, bi) end
[] tryPropag =>  repeat*(dk(me, ipso, bs, ox)) end
[] tryTermin =>  dk(co, di) end
```

Chapter 5

TOM **Technology**

TOM is a language extension which adds new matching primitives to languages like C, Java, and Caml. Although rich and complex, TOM is not a stand-alone language: like a preprocessor, it strongly relies on the underlying language (C, Java, or Caml), called host-language in the following. To this language, TOM adds several constructs. TOM is particularly well-suited for programming various transformations on trees/terms or XML data-structures.

5.1 TOM **Constructs**

A TOM program is a host language program (namely C, Java, or Caml) extended by several new constructs such as %match, %rule, %include, %vas, or backquote. TOM is a multi-languages compiler, so, its syntax depends on the host language syntax.

Basically, a TOM program is list of blocks, where each block is either a TOM construct, or a sequence of characters. The idea is that after transformation, the sequence of characters merged with the compiled TOM constructs should be a valid host language program. In the previous example, %include and %match constructs are replaced by function definitions and Java instructions, making the resulting program a correct Java program.

The types of blocks are the following:

MatchConstruct is translated into a list of instructions. This construct may appear anywhere a list of instructions is valid in the host language.

RuleConstruct is translated into a function definition. This construct may appear anywhere a function declaration is valid in the host language.

BackQuoteTerm is translated into a function call.

IncludeConstruct is replaced by the content of the file referenced by the construct. If the file contains some TOM constructs, they are expanded.

VasConstruct allows to define a Vas grammar. This construct is replaced by the content of the generated mapping.

TypeTerm as well as **Operator**, **OperatorList**, and **OperatorArray** are replaced by some functions definitions.

The main construct, `%match`, is similar to the match primitive found in functional languages: given an object (called subject) and a list of patterns-actions, the match primitive selects the first pattern that matches the subject and performs the associated action. The subject against which we match can be any object, but in practice, this object is usually a tree-based data-structure, also called term in the algebraic programming community.

The match construct may be seen as an extension of the classical switch/case construct. The main difference is that the discrimination occurs on a term and not on atomic values like characters or integers: the patterns are used to discriminate and retrieve information from an algebraic data structure. Therefore, TOM is a good language for programming by pattern matching

A *MatchConstruct* is composed of two parts:

- a list of host language variables called *subjects*. These variables should reference the objects to be matched.

- a list of pairs (pattern, action), where an action is a set of host language instructions which is executed each time a pattern matches the subjects.

For example, the block below will match twice:

```
String t = ''hello'';
%match(String t) {
   (before*,'l',after*) -> {
      System.out.println("we have found " + 'before* +
            " before 'l' and " + 'after* + "after");
   }
}
```

and it will provide the output:

```
we have found he before 'l' and lo after
we have found hel before 'l' and o after
```

5.2 Vas **and** ApiGen

In order to describe user defined abstract data-types, TOM provides a signature definition mechanism (`%typeterm`, `%typelist`, `%op`, etc.). However, TOM does not provide any support to implement these abstract data-types, we can only define algebraic mappings for given Java classes. In order to cope with this lacking, the systems Vas and ApiGen were developed. ApiGen is a system which takes a many-sorted signature as input, and it generates both a concrete implementation for the abstract data-type and a mapping for TOM. Vas is a preprocessor for ApiGen which provides a human-readable syntax definition formalism inspired from SDF. These two systems are useful for manipulating Abstract Syntax Trees since they offer an efficient implementation based on ATerms which supports maximal memory sharing, strong static typing, as well as parsers and pretty-printers. The memory sharing is very important for the implementation of molecules because the terms encoding the molecules have in general many common subterms.

We illustrate the use of Vas considering the following Vas module:

```
module term
imports
```

```
public
  sorts Term

abstract syntax
  a -> Term
  b -> Term
  c -> Term
  f(arg1:Term) -> Term
  g(arg1:Term, arg2:Term) -> Term
```

After running Vas, new directories are generated. They contain all classes that make up the API for the signature. At the root level, this directory contains 6 classes (`AbstractType`, `Factory`, `Fwd`, `FwdVoid`, `VisitableFwd`, and `Visitor`) and a mapping for TOM (term.tom).

Sort classes are stored in the separate directory types. This directory contains abstract base classes for each sort defined in the signature (`Term`), and one subdirectory per sort that contains concrete classes (`term`). Production classes are named according to the operators of the signature. To be more concrete, here is the list of generated files for the given input signature:

```
term
term/term.tom
term/termAbstractList.java
term/termAbstractType.java
term/termFactory.java
term/termFwd.java
term/termFwdVoid.java
term/termVisitableFwd.java
term/termVisitor.java
term/types
term/types/term
term/types/term/A.java
term/types/term/B.java
term/types/term/C.java
term/types/term/F.java
term/types/term/G.java
term/types/Term.java
```

The `Factory` class is used to create objects instances of the operators defined in the signature.

The `AbstractType` class is an abstract base class from which all sort classes of the signature will derive. The `Fwd`, `FwdVoid`, `VisitableFwd`, and `Visitor` classes correspond to visitors that have to be used in conjunction with the JJTraveler or the muTraveler visitor framework.

The Visitor Design Pattern

Objects form a natural way for representing such source models, offering appropriate abstraction mechanisms, classes for organizing model elements, and object manipulation and navigation for operating on the model. In order to implement a range of operations on an object-oriented source model, the *Visitor* design pattern can be used [DV04]. The intent of the visitor design pattern is to "represent an operation to be performed on the elements of an object structure. A visitor lets you define a new operation without changing the classes of the elements on which it operates" [GHJV94]. Often, visitors are constructed to traverse an object structure according to a particular built-in strategy, such as top-down, bottom-up, or breadth-first.

A typical example of the use of the visitor pattern in program understanding tools involves the traversal of abstract syntax trees. The pattern offers an abstract class Visitor, which defines a series of methods that are invoked when nodes of a particular type (expressions, statements, etc.) are visited. A concrete Visitor subclass refines these methods in order to perform specific actions when it gets accepted by a given syntax tree.

Visitors are useful for analysis and manipulation of source models for several reasons. Using visitors makes it easy to traverse structures that consist of many different kinds of nodes, while conducting actions on only a selected number of them. Moreover, visitors make it possible to add new forms of analysis easily, without modifying the class hierarchy representing node types. The implementation of these different analysis can be isolated in individual classes, rather than being scattered over the various node types.

5.3 Strategies

TOM provides a library inspired by ELAN, Stratego, and JJTraveler, which allows to easily define various kind of traversal strategies. **mutraveler.tom** provides an algebraic view of elementary strategy constructors.

The following operators are the key-component that can be used to define more complex strategies. In this framework, the application of a strategy to a term can fail. In Java, the failure is implemented by an exception (`VisitFailure`).

$$
\begin{array}{lll}
Identity@(t) & => & t \\
Fail@(t) & => & failure \\
Sequence(s1,s2)@(t) & => & failure \text{ if } s1@(t) \text{ fails} \\
 & & s2@(t') \text{ if } s1@(t) => t' \\
Choice(s1,s2)@(t) & => & t' \text{ if } s1@(t) => t' \\
 & & s2@(t) \text{ if } s1@(t) \text{ fails} \\
All(s)@(f(t1,...,tn)) & => & f(t1',...,tn') \text{ if } s1@(t1) => t1',...,sn@(tn) => tn' \\
 & & failure \text{ if there exists } i \text{ such that } si@(ti) \text{ fails} \\
All(s)@(cst) & => & cst \\
One(s)@(f(t1,...,tn)) & => & f(t1,...,ti',...,tn) \text{ if } si@(ti) => ti' \\
 & & failure \text{ if } s1@(t1) \text{ fails},...,sn@(tn) \text{ fails} \\
One(s)@(cst) & => & failure \\
Omega(i,s)@(f(t1,...,tn)) & => & f(t1,...,ti',...,tn) \text{ if } si@(ti) => ti' \\
 & & failure \text{ if } si@(ti) \text{ fails}
\end{array}
$$

In order to define recursive strategies, we introduce the *mu* abstractor. This allows to give a name to the current strategy, which can be referenced later.

$$
\begin{array}{lll}
Try(s) & = & Choice(s,Identity) \\
Repeat(s) & = & mu\,x.Choice(Sequence(s,x),Identity()) \\
OnceBottomUp(s) & = & mu\,x.Choice(One(x),s)) \\
BottomUp(s) & = & mu\,x.Sequence(All(x),s)) \\
Innermost(s) & = & mu\,x.Sequence(All(x),Try(Sequence(s,x))))
\end{array}
$$

The *Try* strategy never fails: it tries to apply the strategy *s*. If it succeeds, the result is returned. Otherwise, the *Identity* strategy is applied, and the subject is not modified.

The *Repeat* strategy applies the strategy *s* as many times as possible, until a failure occurs. The last unfailing result is returned.

The strategy *OnceBottomU p* tries to apply the strategy *s* once, starting from the leftmost-innermost leaves. *BottomU p* looks like *OnceBottomU p* but is not similar: *s* is applied to all nodes, starting from the leaves. Note that the application of *s* should not fail, otherwise the whole strategy also fails.

The strategy *Innermost* tries to apply *s* as many times as possible, starting from the leaves. This construct is useful to compute normal forms.

In order to use the runtime library and the JJTraveler framework, we need to import the following packages:

```
import tom.library.strategy.mutraveler.*;
import jjtraveler.reflective.VisitableVisitor;
import jjtraveler.Visitable;
import jjtraveler.VisitFailure;
```

We also need to import he corresponding mapping:

```
%include { mutraveler.tom }
```

Let us consider the Vas signature from section 5.2.

We want to implement the rewriting system given by the following three rewrite rules: $a \rightarrow b$, $b \rightarrow c$, $g(c,c) \rightarrow c$. This is achieved by means of the above Java class:

```
class RewriteSystem extends mypackage.term.termVisitableFwd {
  public RewriteSystem() {
    super('Fail());
  }

  public Term visit_Term(Term arg) throws VisitFailure {
    %match(Term arg) {
      a()        -> { return 'b(); }
      b()        -> { return 'c(); }
      g(c(),c()) -> { return 'c(); }
    }
    return (Term)'Fail().visit(arg);
  }
}
```

We can also use methods to retrieve the visited subterm and its position in the current term, as well as to replace the subterm with another term:

```
public Term visit_Term(Term arg) throws VisitFailure {
  %match(Term arg) {
    a() -> {
      Position pos = MuTraveler.getPosition(this);
      System.out.println("a -> b at " + pos);
      System.out.println(globalSubject + " at " + pos + " = " +
                                pos.getSubterm().visit(globalSubject));
      System.out.println("rwr into: " +
                          pos.getReplace('b()).visit(globalSubject));
      return 'b();
    }
    b() -> {
```

```
      System.out.println("b -> c at " + MuTraveler.getPosition(this));
      return 'c();
    }
  g(c(),c()) -> {
      System.out.println("g(c,c) -> c at " + MuTraveler.getPosition(this));
      return 'c();
    }
  }
  return (Term)'Fail().visit(arg);
}
```

Then, it becomes quite easy to define various strategies on top of this user-defined strategy:

```
Term subject = 'f(g(g(a,b),g(a,a)));
VisitableVisitor rule = new RewriteSystem();
try {
  System.out.println("subject      = " + subject);
  System.out.println("\nonceBottomUp = " +
            MuTraveler.init('OnceBottomUp(rule)).visit(subject));
  System.out.println("\nonceBottomUpId= " +
            MuTraveler.init('OnceBottomUpId(ruleId)).visit(subject));
  System.out.println("\nbottomUp      = " +
            MuTraveler.init('BottomUp(Try(rule))).visit(subject));
  System.out.println("\nbottomUpId    = " +
            MuTraveler.init('BottomUp(ruleId)).visit(subject));
  System.out.println("\ninnermost     = " +
            MuTraveler.init('Innermost(rule)).visit(subject));
  System.out.println("\ninnermostSlow = " +
            MuTraveler.init('Repeat(OnceBottomUp(rule))).visit(subject));
  System.out.println("innermostId   = " +
            MuTraveler.init('InnermostId(ruleId)).visit(subject));
} catch (VisitFailure e) {
  System.out.println("reduction failed on: " + subject);
}
```

The results of rewriting the term f(g(g(a,b),g(a,a))) by means of the three rules using different strategies are the following:

```
subject      = f(g(g(a,b),g(a,a)))

a -> b at [1, 1, 1]
f(g(g(a,b),g(a,a))) at [1, 1, 1] = a
rwr into: f(g(g(b,b),g(a,a)))
onceBottomUp  = f(g(g(b,b),g(a,a)))

a -> b at [1, 1, 1]
onceBottomUpId= f(g(g(b,b),g(a,a)))

a -> b at [1, 1, 1]
f(g(g(a,b),g(a,a))) at [1, 1, 1] = a
rwr into: f(g(g(b,b),g(a,a)))
b -> c at [1, 1, 2]
a -> b at [1, 2, 1]
f(g(g(a,b),g(a,a))) at [1, 2, 1] = a
```

29

```
rwr into: f(g(g(a,b),g(b,a)))
a -> b at [1, 2, 2]
f(g(g(a,b),g(a,a))) at [1, 2, 2] = a
rwr into: f(g(g(a,b),g(a,b)))
bottomUp      = f(g(g(b,c),g(b,b)))

a -> b at [1, 1, 1]
b -> c at [1, 1, 2]
a -> b at [1, 2, 1]
a -> b at [1, 2, 2]
bottomUpId    = f(g(g(b,c),g(b,b)))

a -> b at [1, 1, 1]
f(g(g(a,b),g(a,a))) at [1, 1, 1] = a
rwr into: f(g(g(b,b),g(a,a)))
b -> c at [1, 1, 1]
b -> c at [1, 1, 2]
g(c,c) -> c at [1, 1]
a -> b at [1, 2, 1]
f(g(g(a,b),g(a,a))) at [1, 2, 1] = a
rwr into: f(g(g(a,b),g(b,a)))
b -> c at [1, 2, 1]
a -> b at [1, 2, 2]
f(g(g(a,b),g(a,a))) at [1, 2, 2] = a
rwr into: f(g(g(a,b),g(a,b)))
b -> c at [1, 2, 2]
g(c,c) -> c at [1, 2]
g(c,c) -> c at [1]
innermost     = f(c)

a -> b at [1, 1, 1]
f(g(g(a,b),g(a,a))) at [1, 1, 1] = a
rwr into: f(g(g(b,b),g(a,a)))
b -> c at [1, 1, 1]
b -> c at [1, 1, 2]
g(c,c) -> c at [1, 1]
a -> b at [1, 2, 1]
f(g(g(a,b),g(a,a))) at [1, 2, 1] = a
rwr into: f(g(g(a,b),g(b,a)))
b -> c at [1, 2, 1]
a -> b at [1, 2, 2]
f(g(g(a,b),g(a,a))) at [1, 2, 2] = a
rwr into: f(g(g(a,b),g(a,b)))
b -> c at [1, 2, 2]
g(c,c) -> c at [1, 2]
g(c,c) -> c at [1]
innermostSlow = f(c)

a -> b at [1, 1, 1]
b -> c at [1, 1, 1]
b -> c at [1, 1, 2]
g(c,c) -> c at [1, 1]
```

```
a -> b at [1, 2, 1]
b -> c at [1, 2, 1]
a -> b at [1, 2, 2]
b -> c at [1, 2, 2]
g(c,c) -> c at [1, 2]
g(c,c) -> c at [1]
innermostId   = f(c)
```

```
a -> b at [1, 2, 1]
b -> c at [1, 2, 1]
a -> b at [1, 2, 2]
b -> c at [1, 2, 2]
g(c,c) -> c at [1, 2]
g(c,c) -> c at [1]
innermostId   = f(c)
```

Chapter 6

TOM **Implementation of the Oxidizing Pyrolysis Process**

6.1 The Signature

The signature of the decorated labelled term rewriting system considered in chapter 3 (Figure 3.1) is given by the Vas module data:

```
module data
  imports
  public
  sorts Bond Atom IntList Symbol Radical RadicalList
        Pack PackList BiPack BiPackList MBiPack MBiPackList

  abstract syntax
      none   -> Bond
      simple -> Bond
      double -> Bond
      triple -> Bond
      arom   -> Bond

      C   -> Atom
      arC -> Atom
      O   -> Atom
      H   -> Atom
      e   -> Atom

      concInt( int* ) -> IntList
      symb(atom:Atom, labels:IntList) -> Symbol

      idrad -> Radical
      rad(bond:Bond,symbol:Symbol,radList:RadicalList) -> Radical
      concRad( Radical* ) -> RadicalList

      pack(term:Radical, fcan:str) -> Pack
      concPack( Pack* ) -> PackList

      bipack(pk1:Pack, pk2:Pack) -> BiPack
```

```
concBiPack( BiPack* ) -> BiPackList

mbipack(mult:int, bp:BiPack) -> MBiPack
concMBiPack( MBiPack* ) -> MBiPackList
```

The notation `concRad(Radical*)` means that `concRad` is a variadic associative operator which takes a list of `Radical` sorted terms as arguments, and returns a `RadicalList`.

A decorated labelled tree (with a root) is represented as a decorated term of sort `Radical` as follows:

- a leaf *v* is a term of sort `Radical`,

    `rad(b, symb(a, concInt(labs*)), concRad()),`

    where a encodes the label of the leaf, b encodes the label of the edge connecting *v* with his father, and `labs*` is a possibly empty list of integers representing the associated set of cycle labels;

- an internal vertex is a term of sort `Radical`,

    `rad(b, symb(a, concInt(labs*)), concRad(rads*)),`

    where `rads*` encodes the list of its term represented children;

- the root has a dummy bond label, `none`, in order to make the representation uniform for all vertices.

6.2 Reaction Rules

We present in TOM or Java the ideas behind the implementation of reaction rules.

If we obtain a reaction product having a radical point, then we turn the term/tree in order to have the electron in the root (in a à-la-SMILES representation); this reduces the number of searched patterns.

A reaction product is always represented as a pair composed of a term of sort `Radical` and its canonical form "à-la-SMILES" of sort `str`.

Each of the reactions (ui), (bi), (me), (ipso), (bs), (ox), (di) return a list of pairs of reaction products and the multiplicity of obtaining them from the same reactants (also called *degeneration*) (we need the multiplicity for enumerating the reactions); meanwhile, the reactions (co.O.) and (co) return at most one reaction product.

We illustrate the necessity of the degeneration factor by means of the example below.

Example 6.1 *The four unimolecular initiation reactions on iso-octane by breaking a* **C–C** *bond are the following:*

$$
\begin{array}{ccc}
& \underset{\displaystyle\mathrm{CH_3}\quad\mathrm{CH_3}}{\mathrm{H_3C{-}\underset{\displaystyle\mathrm{CH_3}}{\overset{\displaystyle\mathrm{CH_3}}{\mathrm{C}}}{-}CH_2{-}CH{-}CH_3}} & \longrightarrow \quad \bullet\,\mathrm{CH_3} \;+\; \bullet\,\mathrm{C{-}CH_2{-}CH{-}CH_3} \qquad \textit{[3]} \qquad (6.1)
\end{array}
$$

$$
\begin{array}{c}
\overset{\displaystyle CH_3}{\underset{\displaystyle CH_3}{H_3C-\overset{|}{\underset{|}{C}}-CH_2-\overset{CH_3}{\overset{|}{CH}}-CH_3}} \quad \longrightarrow \quad CH_3-\overset{CH_3}{\overset{|}{C}}\bullet \;+\; \bullet CH_2-\overset{CH_3}{\overset{|}{CH}}-CH_3 \qquad \textit{[1]} \qquad (6.2)
\end{array}
$$

$$
\begin{array}{c}
\overset{\displaystyle CH_3}{\underset{\displaystyle CH_3}{H_3C-\overset{|}{\underset{|}{C}}-CH_2-\overset{CH_3}{\overset{|}{CH}}-CH_3}} \quad \longrightarrow \quad CH_3-\overset{CH_3}{\overset{|}{C}}-\underset{\bullet}{CH_2} \;+\; \bullet CH-CH_3 \qquad \textit{[1]} \qquad (6.3)
\end{array}
$$

$$
\begin{array}{c}
\overset{\displaystyle CH_3}{\underset{\displaystyle CH_3}{H_3C-\overset{|}{\underset{|}{C}}-CH_2-\overset{CH_3}{\overset{|}{CH}}-CH_3}} \quad \longrightarrow \quad CH_3-\overset{CH_3}{\overset{|}{C}}-CH_2-\underset{\bullet}{\overset{CH_3}{\overset{|}{CH}}} \;+\; \bullet CH_3 \qquad \textit{[2]} \qquad (6.4)
\end{array}
$$
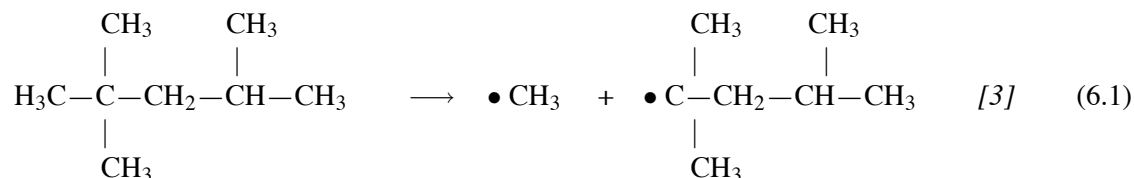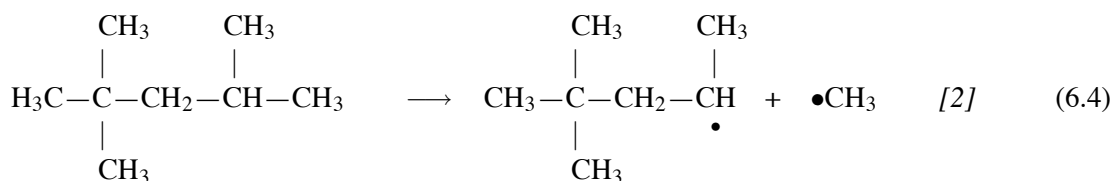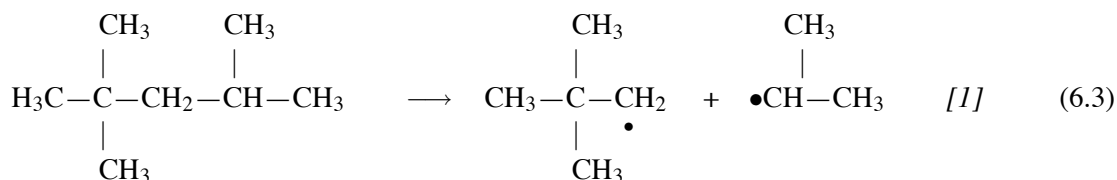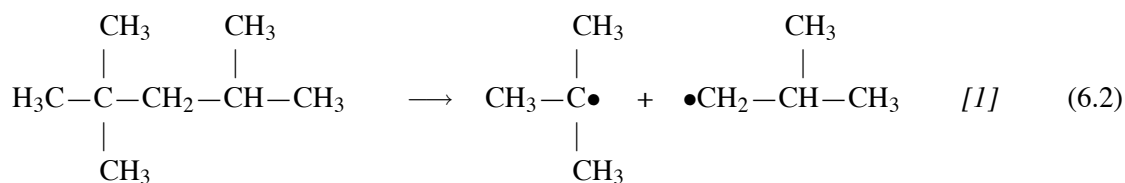
For the iso-octane molecule, there are seven possibilities of **C–C** bond breaking, but, due to the symmetries of the molecule, some reactions are identical. For example, the reaction (6.1) occurs three times. This reactions is written only once, and the number of identical reactions represents the degeneration of the reaction.

Remark 6.1 *The degenerations of the unimolecular initiation reactions for the iso-octane molecule by* **C–C** *bond breaking appear in italics at the end of each reaction from the example 6.1:*

- *the reaction (6.1) has a degeneration of 3;*

- *each of the reactions (6.2) and (6.3) has a degeneration of 1;*

- *the reaction (6.4) has a degeneration of 2: there are two possibilities of breaking the bond* **C–C**.

We call an *insignificant* reactant one of the following molecules or radicals: **O=O** for (bi) and (ox), $\bullet\beta$ for (me), $\bullet$**H** for (ipso), $\bullet$**O**$\bullet$ for (co.O.). In general, the *significant* reactant represents the searched pattern for reactions.

The implementations of the initiation and propagation reaction rules take as input a term of sort `Radical` (corresponding to the significant reactant) and return a list of pairs of `Radical` sorted terms.

The rules from the initiation and propagation stages can be reduced to the form $r : t \to t'_1 + t'_2$, where r is the label of the reaction rule, and t, t'_1, t'_2 are terms of sort *Radical*, except (me) which is parameterized by a β radical. However, if the reaction rule is $r : t \to t'_1$, then we can consider the term t'_2 to be the dummy radical *idrad*(). In general we implement the rule by means of a match construct as below:

- the match argument is the term we want to rewrite (or the significant reactant);

- the pattern is the generic pattern of the reaction rule (as presented in section 2.2.3);

- in the action-block associated to the pattern we combine Java code with TOM constructs or strategies in order to compute the terms $\sigma(t'_1)$ and $\sigma(t'_2)$, where σ is the substitution defined by instantiating the variables from the pattern t.

6.2.1 Unimolecular Initiation

$$x - y \longrightarrow x \bullet + \bullet y$$
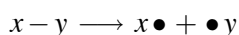
During the unimolecular initiation, a **C–H** bond is broken if the carbon atom is non-aromatic, while a **C–C** bond is broken if it is not a component of a cycle.

We search a non-aromatic carbon atom in the term representation which has at least one hydrogen bound by examining all subterms of sort `Radical`. This is implemented by means of a class `UICHRule` which extends the class `act.data.dataVisitableFwd`.

The result of traversing bottom-up the tree by visiting all subterms of sort `Radical` in the above manner is achieved by means of the following two lines:

```
VisitableVisitor uiCHRule = new UICHRule(resultCol);
MuTraveler.init('BottomUp(Try(uiCHRule))).visit(subject));
```

The searched pattern is the term of sort `Radical`:

```
rad(b, symb(C(), concInt(labs*)), concRad(rads*))
```

satisfying the condition that the number of hydrogen atoms connected to the **C** atom is greater or equal to 1. This number is computed by substracting from the valence of the carbon atom the weight of the bond b and of its bonds with the children from the list of radicals `concRad(rads*)` (if any), as well as the number of cycle labels from the list `concInt(labs*)`.

Considering that the variable `globalSubject` keeps the value of the term participating at the reaction, the function below finds the reaction products and puts them in a collection (in fact a reaction product is always •**H**):

```
public Radical visit_Radical(Radical arg) throws VisitFailure {
  Radical r1, r2;
  %match(Radical arg) {
    rad(b, symb(C(), concInt(labs*)), concRad(rads*)) -> {
      if(nH(arg) >= 1) {
        Position pos = MuTraveler.getPosition(this);
        r1 = insertElectronAsRootForH(
                  (Radical)pos.getSubterm().visit(globalSubject));
        r2 = hangE((Radical)pos.getReplace(r1).visit(globalSubject));

        col.add('bipack(pack(eh, seh),
                    pack(r2, toCanonicSmiles(radical2usmiles(r2))))));
        return arg;
      }
    }
  }
}
```

In the previous piece of code we attach an electron to the found carbon atom, we insert the new term in the context, and then we twist the term by means of `hangE` such that node labelled by `e` becomes the root in the corresponding molecular tree.

In order to break a bond **C–C**, we search a subterm of sort `Radical` with a simple bond and a non-aromatic carbon atom in top. If the atom from the context connected by this very bond with the found carbon atom is also a carbon atom, and the subterm and the context do not share common labels, then the bond break is allowed.

6.2.2 Bimolecular Initiation

$$\mathbf{O} = \mathbf{O} + \mathbf{H} - x \longrightarrow \bullet\mathbf{OOH} + \bullet x$$
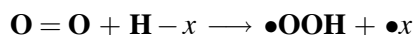
This reaction implementation is similar with the **C–H** break bond, except we obtain a radical •**OOH** instead of •**H**.

6.2.3 Metathesis

$$\bullet\beta + \mathbf{H} - x \longrightarrow \beta - \mathbf{H} + \bullet x$$
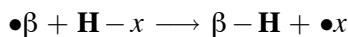
This reaction is also very similar with the unimolecular initiation on a **C–H** bond, except it is parameterized after a β radical. This time the given radical reacts with the β radical, and instead of •**H** we obtain a radical by replacing in the radical β the electron for a hydrogen atom.

6.2.4 Ipso

$$\bullet\mathbf{H} + Ar - x \longrightarrow \mathbf{H} - Ar + \bullet x$$
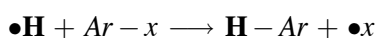
We have two separate traversals of the term for explicit and implicit edges:

explicit edges: we search either a bond **c–C** or a bond **C–c** and we break it; if in the associated graph representation the two atoms are on a cycle, then we find the hidden edge and make it explicit (i.e., we delete the labels encoding the cycle break);

implicit edges: we search a **C** atom with a label; if the same label is found on a **c** atom, then there is an implicit edge between them, and we delete the two labels.

For breaking a bond **C–C** encoded by a pair of labels (therefore corresponding to a implicit edge in the associated molecular graph) the ipso reaction rule is the following:

$$\bullet\mathbf{H} + \mathbf{c}[k] + \mathbf{C}[k] \longrightarrow \mathbf{H} - \mathbf{c} + \bullet\mathbf{C}$$
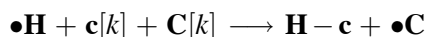
where k denoted the cycle break label.

6.2.5 Beta Scission

$$\bullet x - y - z \longrightarrow x = y + \bullet z$$
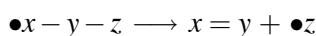
We consider only the cases when the edge between the nodes labelled by x and y respectively is explicit.

The searched pattern (or the left hand side of the rule (bs)) for the case of breaking a **C–H** bond is:

```
rad(none(), symb(e(), concInt()), concRad(
        rad(simple(), symb(x, concInt(labs1*)), concRad(
                   rads1*,
                   rad(simple(), symb(C(), concInt(labs2*)), concRad(rads*)),
                   rads2*))))
```

where x is **C**, **c**, or **O**, and the number of hydrogen bonds for the **C** atom at distance two from **e** is at least 1. In the right side hand of the reaction rule we obtain two reactants:

```
 'rad(none(), symb(x, concInt(labs1*)), concRad(
         rads1*,
         rad(double(), symb(C(), concInt(labs2*)), concRad(rads*)),
         rads2*))
```
(by transforming the simple bond between x and **C** into a double bond and removing the radical point) and •**H**.

The searched pattern (or the left hand side of the rule (bs)) for breaking a **C–C** bond is:

```
rad(none(), symb(e(), concInt()), concRad(
       rad(simple(), symb(x, concInt(labs*)), concRad(
                 rads1*,
                 rad(simple(), symb(y, concInt(labs1*)), concRad(
                       rads3*,
                       rad(simple(), symb(z, concInt(labs2*)), concRad(rads5*)),
                       rads4*)),
                 rads2*))))
```

where x can be **C**, **c**, or **O**, and y and z are not both **c**. The resulted reactants are:

```
 'rad(none(), symb(x, concInt(labs*)), concRad(
         rads1*,
         rad(double(), symb(y, concInt(labs1*)), concRad(rads3*, rads4*)),
         rads2*))
```

and

```
 'rad(none(), symb(e(), concInt()), concRad(
         rad(simple(), symb(z, concInt(labs2*)), concRad(rads5*))))
```

If the nodes corresponding to y and z in the molecular graph are on a cycle, then we find the hidden edge and make it explicit.

These two presented reaction rules correspond to the case when the bond between y and z is not encoded by a pair of labels.

For a reaction (bs) which breaks a bond **C—C** encoded by a pair of labels we consider the following rule:

$$\bullet x - y[k] + z[k] \longrightarrow x = y + \bullet z$$
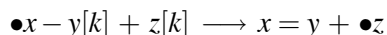
where k denotes the cycle break label. **Y** and **Z** are carbon atoms, but they are not both aromatic. We search the pattern:

```
rad(none(), symb(e(), concInt()), concRad(
       rad(simple(), symb(x, concInt(labs*)), concRad(
                 rads1*,
                 rad(simple(), symb(y, concInt(l1*, k, l2*)), concRad(rads3*)),
                 rads2*))))
```

and then we search for the atom z with the other label k. If y and z are carbon atoms but they are not both aromatic atoms, then we deleted the two labels k, the bond between x and y becomes double, and we attach an **e** atom to z. We obtain only one radical with the electron on top and its only child the atom z.

There is second reaction rule for beta scission:

$$\bullet x = y - z \longrightarrow x\#y + \bullet z$$

which transforms the double bond into a triple bond. This rule is easily implemented using the same methodology as of the first rule.

6.2.6 Oxidation

$$O = O + H - x - y\bullet \longrightarrow \bullet OOH + x = y$$

For this reaction as well for combination with $\bullet O\bullet$, beta scission for **C–H** and explicit **C–C** bonds, combination, and disproportionation, we make extensive use of the rooted-position of the electron in a radical.

We try to match a given term of sort `Radical` with the term:

```
rad(none(), symb(e(), concInt()), concRad(
        rad(simple(), symb(y, concInt(labs1*)), concRad(
            rads1*,
            rad(simple(), symb(C(), concInt(labs2*)), concRad(rads*)),
            rads2*))))
```
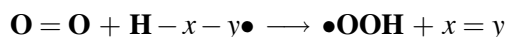
such that y is a carbon atom, aromatic or not aromatic, and the carbon situated at distance two from the electron has at least one hydrogen atom. The significant reaction product is given by the following term:

```
'rad(none(), symb(y, concInt(labs1*)), concRad(
            rads1*,
            rad(double() , symb(C(), concInt(labs2*)), concRad(rads*)),
            rads2*))
```

As for (bs), we have also implemented a second reaction rule for oxidation:

$$O = O + H - x = y\bullet \longrightarrow \bullet OOH + x\#y$$

6.2.7 Combination with $\bullet O\bullet$

$$\bullet O\bullet + \bullet x \longrightarrow \bullet O - x$$
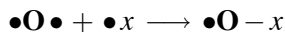
The implementation for this reaction rule is very simple and it looks very much like a classical conditional rewrite rule because both sides of the rule consist in one reactant:

```
%match(Radical subject) {
   rad(none(), symb(e(), concInt()), concRad(
            rad(simple(), symb(x, concInt(labs*)), concRad(rads*)))) -> {
     if(('x == 'H()) || ('x == 'C())) {
       radical = 'rad(none(), symb(e(), concInt()), concRad(
                   rad(simple(), symb(O(), concInt()), concRad(
                     rad(simple(), symb(x, concInt(labs*)), concRad(rads*))))));
          return 'pack(radical, toCanonicSmiles(radical2usmiles(radical)));
      }
   }
}
```

6.2.8 Combination

$$\bullet x + \bullet y \longrightarrow x - y$$
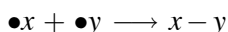
A β radical and a Y radical are combined by creating a simple bond between the two subterms of sort `Radical` children of the electron atoms:

```
%match(Radical r1, Radical r2) {
  rad(none(), symb(e(), concInt()), concRad(
            rad(simple(), symb(x, concInt(labs1*)), concRad(rads1*)))),
  rad(none(), symb(e(), concInt()), concRad(
            rad(simple(), symb(y, concInt(labs2*)), concRad(rads2*)))) -> {
      radical = 'rad(none(), symb(x, concInt(labs1*)), concRad(
                    rads1*,
                    rad(simple(), symb(y, concInt(labs2*)), concRad(rads2*))));
      return 'pack(radical, toCanonicSmiles(radical2usmiles(radical)));
  }
}
```

6.2.9 Disproportionation

$$\bullet x + \mathbf{H} - y - z\bullet \longrightarrow x - \mathbf{H} + y = z$$
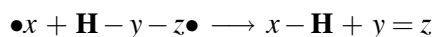
This case is not symmetric as combination, therefore, given two radicals r1 and r2, we must consider the disproportionation of r1 with r2, as well as the disproportionation of r2 with r1. The main chemical filter for disproportionation consists in restricting the free radical $\bullet x$ to the class of β radicals, and the free radical $\mathbf{H} - y - z\bullet$ to the class of Y radicals.

The searched pattern for $\bullet x$ is :

```
  rad(none(), symb(e(), concInt()), concRad(
            rad(simple(), symb(x, concInt(labs1*)), concRad(rads*))))
```

while the pattern for $\mathbf{H} - y - z\bullet$ is:

```
  rad(none(), symb(e(), concInt()), concRad(
            rad(simple(), symb(z, concInt(labs2*)), concRad(
                rads1*,
                rad(simple(), symb(y, concInt(labs3*)), concRad(rads3*)),
                rads2*))))
```

satisfying the conditions: x is a non-aromatic atom, and y and z are wether $\mathbf{C}$ or $\mathbf{O}$.

The resulted molecule $x - \mathbf{H}$ translates in TOM into:

```
    'rad(none(), symb(H(), concInt()), concRad(
                rad(simple(), symb(H(), concInt()), concRad())))
```

if x is $\mathbf{H}$, and into

```
    'rad(none(), symb(x, concInt(labs1*)), concRad(rads*))
```

otherwise.

The other resulted molecule, $y = z$, is encoded in TOM as:

```
    'rad(none(), symb(z, concInt(labs2*)), concRad(
                rads1*,
                rad(double(), symb(y, concInt(labs3*)), concRad(rads3*)),
                rads2*))
```

6.3 Simulation of the Reactor Dynamics

6.3.1 The GasEl Approach in TOM

The semantics of the reactor follows closely the algorithms given in section 2.2.4.

Even if for implementing a reaction rule we try to match only the "significant" reactant, the implementation of the reactor assumes that the insignificant reactants are also in the soup in order to be able to apply a reaction rule. For example, the bimolecular initiation rule cannot take place on a molecule with a **C–H** bond if the molecule **O=O** is not in the soup. Therefore the existence in the soup of "insignificant" reactants represents a precondition/guard for applying a reaction rule.

Initiation Stage

Input: a PackList consisting of the given set of reactants and their canonical "à-la-SMILES" representation
Output: a PackList of reactants resulted from applying *ui* and *bi*.

```
 public PackList initiation(PackList pl, FileWriter rout) throws IOException {
   MBiPackList mbpl;
   Iterator it;
   Pack pk;
   Radical beta;
   PackList plist1 = pl, plist2 = 'concPack();

   %match(PackList plist1) {
     concPack(_*, pack(rad1, fc1), _*) -> {
        if( !isFreeRadical('rad1) ) {
           mbpl = ui_CC('rad1);
           if( mbpl != 'concMBiPack() ) {
              writeReactions(rout, "uiCC", 'rad1, mbpl);
              plist2 = addMBiPackList2PackList(mbpl, plist2);
           }
           mbpl = ui_CH('rad1);
           if( mbpl != 'concMBiPack() ) {
              writeReactions(rout, "uiCH", 'rad1, mbpl);
              plist2 = addMBiPackList2PackList(mbpl, plist2);
           }
           if( avail(ctRads.soo, plist1) ) {
              mbpl = bi('rad1);
              if( mbpl != 'concMBiPack() ) {
                 writeReactions(rout, "  bi", ctRads.oo, 'rad1, mbpl);
                 plist2 = addMBiPackList2PackList(mbpl, plist2);
              }
           }
        }
     }
   }
   return appendPackLists(plist1, plist2);
 }
```

Given the list of reactants, we apply a reaction on a reactant only if it is not a free radical. Then if a reaction type is applicable (i.e., the resulted list of reactants is not empty), the reactions are written

in a file, and the list of resulted reactants is added to the list of existing reactants. The reaction (bi) takes place only if the molecule **O–O** is available in the input list of reactants.

In the end the list if initial reactants concatenated with the list of resulted reactants is returned.

In fact the initiation stage consists of applying only the unimolecular and bimolecular initiations on molecules; but we can consider also the metathesis and ipso reactions in order to stay close to the GasEl implementation. For fear that radicals resulted from metathesis and ipso applications become subjects for the unimolecular and bimolecular initiation reaction rules, we apply (ui) and (bi) on the initial list of molecules, and then we try to apply (me) and (ipso) repeatedly.

Propagation

Input: a PackList consisting of the set of reactants and their canonical "à-la-SMILES" representation resulted from the initialization stage.
Output: a PackList of reactants resulted from applying metathesis, ipso, beta scission, oxidation, and combination with •O• as long as new radicals are obtained.

We control the resulted radicals during the propagation by means of three lists plist1, plist2, plist3:

```
PackList plist1 = 'concPack();
PackList plist2 = pl;
PackList plist3 = plist1;
```

where pl is the list given as input for this stage.

At the beginning of each loop we update the three lists as follows:

plist1 the radicals we are applying the rules on, i.e. the value of plist2;

plist2 the empty list; at the end of each loop block plist2 contains only the radicals resulted from applying propagation rules on plist1 that are not already in plist1, therefore only new radicals;

plist3 the list of new radicals, plist2, concatenated with the list of old radicals, plist3.

The while loop ends when no new radicals are obtained, therefore when plist2 is empty at the end of a loop block. The output of the propagation stage is provided by plist3.

```
do {
    plist3 = appendPackLists(plist3, plist2);
    plist1 = plist2;
    plist2 = 'concPack();
    %match(PackList plist1) {
        concPack(_*, pack(rad1, fc1), _*) -> {
            /*
             * applying reactions and inserting in plist2 the reaction products
             */
        }
    } // end: match
    plist2 = diff(plist2, plist3);
} // while there are new radicals
while(plist2 != 'concPack());
```

The propagation stage returns `plist3` containing the initial list of reactants, as well as all reaction products obtained during the propagation stage.

By considering the metathesis and ipso reaction rules to be applied also during the initiation stage, we can split the propagation stage in two: *propagation-0* consisting of the reaction rules beta scission, oxidation, and combination with •O•, and *propagation-1* consisting of all five propagation reaction rules.

Termination

Input: a PackList consisting of the set of reactants and their canonical "à-la-SMILES" representation resulted from the propagation stage.
Output: a PackList of reactants resulted from applying combination and disproportionation reaction rules.

The implementation for this stage is similar with one for the initiation stage

The Primary Mechanism

The output list of the termination stage is exactly the output of the reactor taking as input the input list for initiation stage.

Therefore the result given by the reactor for the input `pl` is:

```
termination(propagation(initiation(pl, iout), pout), tout)
```

where `iout`, `pout`, and `tout` are the files where the reactions for each stage respectively are written.

The validation of the process is facilitated by means of examining all reactions occurred during each of the three stages.

6.3.2   The Multiset Rewriting Approach

We add sorts and appropriate constructors for allowing the use of multisets of reactants:

```
sorts MPack MPackList
...
abstract syntax
...
   mpack(mult:int, term:Radical, fcan:str) -> MPack
   concMPack( MPack* ) -> MPackList
```

This approach follows more closely the general algorithm 2.1 for artificial chemistry because we remove the reactants for a successful reaction, whereas for the GasEl approach we leave them in the chemical soup.

The differences between the two approaches occur only in at the description of the reactor dynamics.

The three algorithms corresponding to the three stages have a common part parameterized after a set of reaction rules and an input chemical soup:

$$P(S) \ \mathsf{mUNIT} \ (R : P(R), P_1 : P(S))$$

begin

$P_2 := \emptyset;$

while$(\neg terminate())$ **do**

$(m_1, m_2) := select(P_1);$

for all $(m_1 + m_2 \rightarrow m'_1 + m'_2) \in R$

$P_2 := remove(P_2, m_1, m_2);$

$P_2 := insert(P_2, m'_1, m'_2);$

fi

od

return P_2

end

Now the algorithms for the three stages are:

$P(S) \ \mathsf{mAlgInit} \ (P_0 : P(S))$ $P(S) \ \mathsf{mAlgPropag} \ (P_0 : P(S))$

begin   **begin**

  return $\mathsf{mUNIT}(R_{init}, P_0)$ $i := 0;$

end   **repeat**

                                           $P_{i+1} := \mathsf{mUNIT}(R_{propag}, P_i)$

$P(S) \ \mathsf{mAlgTermin} \ (P_0 : P(S))$ $i := i + 1;$

begin   **until** $P_i = P_{i-1};$

  return $\mathsf{mUNIT}(R_{termin}, P_0)$ **return** $P_i;$

end   **end**

The function $remove(P, m_1, m_2)$ decrements the multiplicities of m_1 and m_2 in the multiset P. When a radical's multiplicity reaches 0, then the radical is deleted from the soup.

Different results obtained from a reaction are not put together, but they lead to different states of the system; hence the explosion of the number of states.

The implementation of the three stages is similar with the implementation for the GasEl approach if we consider the differences in the algorithms; the main difference is that we consider a collection of all possible states at each moment.

In this framework we can talk about parallelism because the resources (radicals) are consumed by means of reactions rules. In fact we deal with a maximal parallelism: all rules that can be applied during a stage, must be applied in parallel at the same time.

We can say that the number or reached states can become too big, therefore re-thinking (or optimizing) the implementation of the reactor could be a good idea.

6.3.3   Comparison between the Two Approaches

In the GasEl approach the quantity of a certain type of reactants is not important, hence not represented; we focus on the types of reactants. In the multiset approach the quantity of a certain type of molecule/radical is important and it is represented as its multiplicity. Therefore the GasEl approach focuses on the *qualitative* aspect of the problem, whereas the multiset approach focuses on the *quantitative* aspect.

In the multiset approach we might not obtain all possible results; due to limited quantities of reactants certain classes of reactants disappear (are completely consumed), hence some reactions possible in the GasEl approach are no longer possible here, leading to qualitative differences.

6.4   Comparison between the TOM Implementation and GasEl

In both implementations the syntax is inspired by the SMILES notation. But whereas ELAN allows order-sorted signatures, in Vas only multi-sorted signatures are possible. To cope with this we need to define explicit inclusion operators, and operators in Vas are all in prefix notation.

In ELAN a named rewriting rule can be applied only at the top of a term. Therefore in GasEl for each reactant `AllVisions` is computed: starting from the associated molecular graph, a molecular tree is obtained by choosing a set of edges to cut, and then a *vision* is obtained by choosing a root for the tree. While using TOM we can perform term traversal and change the term not only at its top.

In GasEl the term encoding a free radical is always considered to have the electron as child of the atom in the root. Whereas we always put the electron directly in the root.

The implementation of the reactor dynamics in GasEl uses strategies, while in our implementation we use Java with some TOM constructs and strategies.

Using TOM we benefit from the imperative fetures provided by the host language, Java, and from the efficient implementation based on ATerms by supporting maximal memory sharing.

Chapter 7

Conclusion

The objective of this internship was to give an implementation in TOM of the molecular graph rewriting relation from [BIK05] in the framework of the oxidizing pirolysis mechanism, and to emphasize the capabilities of TOM for modeling this problem.

This report presents an implementation in TOM of the oxidizing pirolysis mechanism, some choices for the model, and the proof for the termination property of the reactor algorithm.

The implementation of the some reaction rules is different and more efficient than the one given in GasEl due to the use of traversal strategies for searching a pattern in a tree-like representation of a reactant. As future work it is interesting to use TOM constructs and strategies as much as possible. A more efficient way of implementing molecular graphs and operations on them seems reasonable. Also as future work, we consider implementing an interpretor for the SMILES notation for molecules.

In appendix we present the mechanism for two types of molecules: iso-octane and ethylcyclohexane. We notice that we obtain more reactions and products than with GasEl, and this is due to a slightly different structure of the reactor.

Bibliography

[BCC+03] Olivier Bournez, Guy-Marie Côme, Valérie Conraud, Hélène Kirchner, and Liliana Ibănescu. A Rule-Based Approach for Automated Generation of Kinetic Chemical Mechanisms. In Robert Nieuwenhuis, editor, *Proceedings of the 14th International Conference on Rewriting Techniques and Applications, RTA 2003, Valencia, Spain, June 9-11, 2003*, volume 2706 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2003.

[BIK05] Olivier Bournez, Liliana Ibanescu, and Hélène Kirchner. From Chemical Rules to Term Rewriting. In *6th International Workshop on Rule-Based Programming*, Nara, Japan, April 2005.

[BKK+98] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An Overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the Second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume15.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science.

[Côm01] Guy-Marie Côme. *Gas-Phase Thermal Reactions. Chemical Engineering Kinetics*. Kluwer Academic Publishers, 2001.

[DZB01] Peter Dittrich, Jens Ziegler and Wolfgang Banzhaf. Artificial Chemistries - A Review. *Artificial Life*, 7(3):225–275, 2001.

[DU73] James Dugundji and Ivar Ugi. An Algebraic Model of Constitutional Chemistry as a Basis for Chemical Computer Programs. *Topics in Current Chemistry*, 39:19–64, 1973.

[DV04] Arie van Deursen and Joost Visser. Source model analysis using the JJTraveler visitor combinator framework Software – Practice & Experience, vol. 34(14):1345–1379, 2004.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[Iban04] Liliana Ibanescu. *Programmation par regles et strategies pour la generation automatique de mecanismes de combustion d'hydrocarbures polycycliques*. These de Doctorat d'Universite, Institut National Polytechnique de Lorraine, Nancy, France, June 2004. `http://www.loria.fr/˜ibanescu/these_en.html`

[TD] TOM Documentation `http://tom.loria.fr`

[WBLF+00] Valérie Warth, Frédérique Battin-Leclerc, René Fournet, Pierre-Alexandre Glaude, Guy-Marie Côme, and Gérard Scacchi. Computer Based Generation of Reaction Mechanisms for Gas-Phase Oxidation. *Computers and Chemistry*, 24:541–560, 2000.

[WWW89] David Weininger, Arthur Weininger, and Joseph L. Weininger. SMILES. 2. Algorithm for Generation of Unique SMILES Notation. *Journal of Chemical Information and Computer Science*, 29:97–101, 1989.

[US] U.S. Environmental Protection Agency: Mid-Continent Ecology Division (Med)- SMILES Tutorial. http://www.epa.gov/med/Prods_Pubs/smiles.htm

Appendix A

Small User's Guide

Input: a list of `Radical` sorted terms

Output: a file with the reactions and a file with the chemical soups after each stage

The sources are placed in a package `act` included in the TOM examples:

```
examples/act/data.vas
examples/act/Chem.t
```

In order to compile and execute this application we do the following:

```
$ ./build.sh act
$ cd build
$ java act.Chem aauniq.exe
```

where `aauniq.exe` is the binary code for the Unique Smiles algorithm.

Appendix B

Generated Mechanisms

B.1   The Mechanism for Iso-octane

Input:

```
1 C(C)(C)(C)(C(C(C)(C)))
1 O(=O)
```

Initiation stage:

```
uiCC) [3] * C(C)(C)(C)(C(C(C)(C))) => e(C) + e(C(C)(C)(C(C(C)(C))))
uiCC) [2] * C(C)(C)(C)(C(C(C)(C))) => e(C) + e(C(C)(C(C(C)(C)(C))))
uiCC) [1] * C(C)(C)(C)(C(C(C)(C))) => e(C(C)(C)) + e(C(C(C)(C)(C)))
uiCC) [1] * C(C)(C)(C)(C(C(C)(C))) => e(C(C(C)(C))) + e(C(C)(C)(C))
uiCH) [9] * C(C)(C)(C)(C(C(C)(C))) => e(H) + e(C(C(C)(C)(C(C(C)(C)))))
uiCH) [6] * C(C)(C)(C)(C(C(C)(C))) => e(H) + e(C(C(C)(C(C(C)(C)(C)))))
uiCH) [1] * C(C)(C)(C)(C(C(C)(C))) => e(H) + e(C(C)(C)(C(C(C)(C)(C))))
uiCH) [2] * C(C)(C)(C)(C(C(C)(C))) => e(H) + e(C(C(C)(C))(C(C)(C)(C)))
  bi) [9] * O(=O) + C(C)(C)(C)(C(C(C)(C))) => e(O(O)) + e(C(C(C)(C)(C(C(C)(C)))))
  bi) [6] * O(=O) + C(C)(C)(C)(C(C(C)(C))) => e(O(O)) + e(C(C(C)(C(C(C)(C)(C)))))
  bi) [1] * O(=O) + C(C)(C)(C)(C(C(C)(C))) => e(O(O)) + e(C(C)(C)(C(C(C)(C)(C))))
  bi) [2] * O(=O) + C(C)(C)(C)(C(C(C)(C))) => e(O(O)) + e(C(C(C)(C))(C(C)(C)(C)))
```

Propagation stage:

```
  me) [9] * e(C) + C(C)(C)(C)(C(C(C)(C))) => C + e(C(C(C)(C)(C(C(C)(C)))))
  me) [6] * e(C) + C(C)(C)(C)(C(C(C)(C))) => C + e(C(C(C)(C(C(C)(C)(C)))))
  me) [1] * e(C) + C(C)(C)(C)(C(C(C)(C))) => C + e(C(C)(C)(C(C(C)(C)(C))))
  me) [2] * e(C) + C(C)(C)(C)(C(C(C)(C))) => C + e(C(C(C)(C))(C(C)(C)(C)))
  me) [9] * e(H) + C(C)(C)(C)(C(C(C)(C))) => H(H) + e(C(C(C)(C)(C(C(C)(C)))))
  me) [6] * e(H) + C(C)(C)(C)(C(C(C)(C))) => H(H) + e(C(C(C)(C(C(C)(C)(C)))))
  me) [1] * e(H) + C(C)(C)(C)(C(C(C)(C))) => H(H) + e(C(C)(C)(C(C(C)(C)(C))))
  me) [2] * e(H) + C(C)(C)(C)(C(C(C)(C))) => H(H) + e(C(C(C)(C))(C(C)(C)(C)))
  me) [9] * e(O(O)) + C(C)(C)(C)(C(C(C)(C))) => O(O) + e(C(C(C)(C)(C(C(C)(C)))))
  me) [6] * e(O(O)) + C(C)(C)(C)(C(C(C)(C))) => O(O) + e(C(C(C)(C(C(C)(C)(C)))))
  me) [1] * e(O(O)) + C(C)(C)(C)(C(C(C)(C))) => O(O) + e(C(C)(C)(C(C(C)(C)(C))))
  me) [2] * e(O(O)) + C(C)(C)(C)(C(C(C)(C))) => O(O) + e(C(C(C)(C))(C(C)(C)(C)))
  me) [2] * e(C) + C(=C)(C)(C(C(C)(C))) => C + e(C(=C(C)(C(C(C)(C)))))
```

```
me) [3] * e(C) + C(=C)(C)(C(C(C)(C))) => C + e(C(C(=C)(C(C(C)(C)))))
me) [6] * e(C) + C(=C)(C)(C(C(C)(C))) => C + e(C(C(C)(C(C(=C)(C)))))
me) [1] * e(C) + C(=C)(C)(C(C(C)(C))) => C + e(C(C)(C)(C(C(=C)(C))))
me) [2] * e(C) + C(=C)(C)(C(C(C)(C))) => C + e(C(C(C)(C))(C(=C)(C)))
me) [6] * e(C) + C(C)(C)(=C(C(C)(C))) => C + e(C(C(C)(=C(C(C)(C)))))
me) [6] * e(C) + C(C)(C)(=C(C(C)(C))) => C + e(C(C(C)(C(=C(C)(C)))))
me) [1] * e(C) + C(C)(C)(=C(C(C)(C))) => C + e(C(C)(C)(C(=C(C)(C))))
me) [1] * e(C) + C(C)(C)(=C(C(C)(C))) => C + e(C(C(C)(C))(=C(C)(C)))
me) [6] * e(C) + C(C)(C)(=C) => C + e(C(C(C)(=C)))
me) [2] * e(C) + C(C)(C)(=C) => C + e(C(=C(C)(C)))
me) [2] * e(C) + C(=C)(C(C(C)(C)(C))) => C + e(C(=C(C(C(C)(C)(C)))))
me) [9] * e(C) + C(=C)(C(C(C)(C)(C))) => C + e(C(C(C)(C)(C(C(=C)))))
me) [2] * e(C) + C(=C)(C(C(C)(C)(C))) => C + e(C(C(C)(C)(C))(C(=C)))
me) [1] * e(C) + C(=C)(C(C(C)(C)(C))) => C + e(C(=C)(C(C(C)(C)(C))))
me) [3] * e(C) + C(C)(=C(C(C)(C)(C))) => C + e(C(C(=C(C(C)(C)(C)))))
me) [9] * e(C) + C(C)(=C(C(C)(C)(C))) => C + e(C(C(C)(C)(C(=C(C)))))
me) [1] * e(C) + C(C)(=C(C(C)(C)(C))) => C + e(C(C(C)(C)(C))(=C(C)))
me) [1] * e(C) + C(C)(=C(C(C)(C)(C))) => C + e(C(C)(=C(C(C)(C)(C))))
me) [3] * e(C) + C(C)(=C) => C + e(C(C(=C)))
me) [2] * e(C) + C(C)(=C) => C + e(C(=C(C)))
me) [1] * e(C) + C(C)(=C) => C + e(C(C)(=C))
me) [3] * e(C) + C(=C(C)(C(C(C)(C)(C)))) => C + e(C(C(C(C(C)(C)(C)))(=C)))
me) [9] * e(C) + C(=C(C)(C(C(C)(C)(C)))) => C + e(C(C(C)(C)(C(C(C)(=C)))))
me) [2] * e(C) + C(=C(C)(C(C(C)(C)(C)))) => C + e(C(C(C)(C)(C))(C(C)(=C)))
me) [2] * e(C) + C(=C(C)(C(C(C)(C)(C)))) => C + e(C(=C(C)(C(C(C)(C)(C)))))
me) [6] * e(C) + C(C)(C)(=C(C(C)(C)(C))) => C + e(C(C(C)(=C(C(C)(C)(C)))))
me) [9] * e(C) + C(C)(C)(=C(C(C)(C)(C))) => C + e(C(C(C)(C)(C(=C(C)(C)))))
me) [1] * e(C) + C(C)(C)(=C(C(C)(C)(C))) => C + e(C(C(C)(C)(C))(=C(C)(C)))
me) [2] * e(H) + C(=C)(C)(C(C(C)(C))) => H(H) + e(C(=C)(C(C(C)(C)))))
me) [3] * e(H) + C(=C)(C)(C(C(C)(C))) => H(H) + e(C(C(=C)(C(C(C)(C)))))
me) [6] * e(H) + C(=C)(C)(C(C(C)(C))) => H(H) + e(C(C(C)(C(C(=C)(C)))))
me) [1] * e(H) + C(=C)(C)(C(C(C)(C))) => H(H) + e(C(C)(C)(C(C(=C)(C))))
me) [2] * e(H) + C(=C)(C)(C(C(C)(C))) => H(H) + e(C(C(C)(C))(C(=C)(C)))
me) [6] * e(H) + C(C)(C)(=C(C(C)(C))) => H(H) + e(C(C(C)(=C(C(C)(C)))))
me) [6] * e(H) + C(C)(C)(=C(C(C)(C))) => H(H) + e(C(C(C)(C(=C(C)(C)))))
me) [1] * e(H) + C(C)(C)(=C(C(C)(C))) => H(H) + e(C(C)(C)(C(=C(C)(C))))
me) [1] * e(H) + C(C)(C)(=C(C(C)(C))) => H(H) + e(C(C(C)(C))(=C(C)(C)))
me) [6] * e(H) + C(C)(C)(=C) => H(H) + e(C(C(C)(=C)))
me) [2] * e(H) + C(C)(C)(=C) => H(H) + e(C(=C(C)(C)))
me) [2] * e(H) + C(=C)(C(C(C)(C)(C))) => H(H) + e(C(=C(C(C(C)(C)(C)))))
me) [9] * e(H) + C(=C)(C(C(C)(C)(C))) => H(H) + e(C(C(C)(C)(C(C(=C)))))
me) [2] * e(H) + C(=C)(C(C(C)(C)(C))) => H(H) + e(C(C(C)(C)(C))(C(=C)))
me) [1] * e(H) + C(=C)(C(C(C)(C)(C))) => H(H) + e(C(=C)(C(C(C)(C)(C))))
me) [3] * e(H) + C(C)(=C(C(C)(C)(C))) => H(H) + e(C(C(=C(C(C)(C)(C)))))
me) [9] * e(H) + C(C)(=C(C(C)(C)(C))) => H(H) + e(C(C(C)(C)(C(=C(C)))))
me) [1] * e(H) + C(C)(=C(C(C)(C)(C))) => H(H) + e(C(C(C)(C)(C))(=C(C)))
me) [1] * e(H) + C(C)(=C(C(C)(C)(C))) => H(H) + e(C(C)(=C(C(C)(C)(C))))
me) [3] * e(H) + C(C)(=C) => H(H) + e(C(C(=C)))
me) [2] * e(H) + C(C)(=C) => H(H) + e(C(=C(C)))
me) [1] * e(H) + C(C)(=C) => H(H) + e(C(C)(=C))
me) [3] * e(H) + C(=C(C)(C(C(C)(C)(C)))) => H(H) + e(C(C(C(C(C)(C)(C)))(=C)))
me) [9] * e(H) + C(=C(C)(C(C(C)(C)(C)))) => H(H) + e(C(C(C)(C)(C(C(C)(=C)))))
```

50

```
  me) [2] * e(H) + C(=C(C)(C(C(C)(C)(C)))) => H(H) + e(C(C(C)(C)(C))(C(C)(=C)))
  me) [2] * e(H) + C(=C(C)(C(C(C)(C)(C)))) => H(H) + e(C(=C(C)(C(C(C)(C)(C)))))
  me) [6] * e(H) + C(C)(C)(=C(C(C)(C)(C))) => H(H) + e(C(C(C)(=C(C(C)(C)(C)))))
  me) [9] * e(H) + C(C)(C)(=C(C(C)(C)(C))) => H(H) + e(C(C(C)(C)(C(=C(C)(C)))))
  me) [1] * e(H) + C(C)(C)(=C(C(C)(C)(C))) => H(H) + e(C(C(C)(C)(C))(=C(C)(C)))
  me) [2] * e(O(O)) + C(=C)(C)(C(C(C)(C))) => O(O) + e(C(=C(C)(C(C(C)(C)))))
  me) [3] * e(O(O)) + C(=C)(C)(C(C(C)(C))) => O(O) + e(C(C(=C)(C(C(C)(C)))))
  me) [6] * e(O(O)) + C(=C)(C)(C(C(C)(C))) => O(O) + e(C(C(C)(C(C(=C)(C)))))
  me) [1] * e(O(O)) + C(=C)(C)(C(C(C)(C))) => O(O) + e(C(C)(C)(C(C(=C)(C))))
  me) [2] * e(O(O)) + C(=C)(C)(C(C(C)(C))) => O(O) + e(C(C)(C))(C(=C)(C)))
  me) [6] * e(O(O)) + C(C)(C)(=C(C(C)(C))) => O(O) + e(C(C(C)(=C(C(C)(C)))))
  me) [6] * e(O(O)) + C(C)(C)(=C(C(C)(C))) => O(O) + e(C(C(C)(C(C(=C)(C)))))
  me) [1] * e(O(O)) + C(C)(C)(=C(C(C)(C))) => O(O) + e(C(C)(C)(C(C(=C)(C))))
  me) [1] * e(O(O)) + C(C)(C)(=C(C(C)(C))) => O(O) + e(C(C)(C))(=C(C)(C)))
  me) [6] * e(O(O)) + C(C)(C)(=C) => O(O) + e(C(C(C)(=C)))
  me) [2] * e(O(O)) + C(C)(C)(=C) => O(O) + e(C(=C(C)(C)))
  me) [2] * e(O(O)) + C(=C)(C(C(C)(C)(C))) => O(O) + e(C(=C(C(C(C)(C)(C)))))
  me) [9] * e(O(O)) + C(=C)(C(C(C)(C)(C))) => O(O) + e(C(C(C)(C)(C(C(=C)))))
  me) [2] * e(O(O)) + C(=C)(C(C(C)(C)(C))) => O(O) + e(C(C(C)(C)(C))(C(=C)))
  me) [1] * e(O(O)) + C(=C)(C(C(C)(C)(C))) => O(O) + e(C(=C)(C(C(C)(C)(C))))
  me) [3] * e(O(O)) + C(C)(=C(C(C)(C)(C))) => O(O) + e(C(C(=C(C(C)(C)(C)))))
  me) [9] * e(O(O)) + C(C)(=C(C(C)(C)(C))) => O(O) + e(C(C(C)(C)(C(=C(C)))))
  me) [1] * e(O(O)) + C(C)(=C(C(C)(C)(C))) => O(O) + e(C(C(C)(C)(C))(=C(C)))
  me) [1] * e(O(O)) + C(C)(=C(C(C)(C)(C))) => O(O) + e(C(C)(=C(C(C)(C)(C))))
  me) [3] * e(O(O)) + C(C)(=C) => O(O) + e(C(C(=C)))
  me) [2] * e(O(O)) + C(C)(=C) => O(O) + e(C(=C(C)))
  me) [1] * e(O(O)) + C(C)(=C) => O(O) + e(C(C)(=C))
  me) [3] * e(O(O)) + C(=C(C)(C(C(C)(C)(C)))) => O(O) + e(C(C(C(C(C)(C)(C)))(=C)))
  me) [9] * e(O(O)) + C(=C(C)(C(C(C)(C)(C)))) => O(O) + e(C(C(C)(C)(C(C(C)(=C)))))
  me) [2] * e(O(O)) + C(=C(C)(C(C(C)(C)(C)))) => O(O) + e(C(C(C)(C)(C))(C(C)(=C)))
  me) [2] * e(O(O)) + C(=C(C)(C(C(C)(C)(C)))) => O(O) + e(C(=C(C)(C(C(C)(C)(C)))))
  me) [6] * e(O(O)) + C(C)(C)(=C(C(C)(C)(C))) => O(O) + e(C(C(C)(=C(C(C)(C)(C)))))
  me) [9] * e(O(O)) + C(C)(C)(=C(C(C)(C)(C))) => O(O) + e(C(C(C)(C)(C(=C(C)(C)))))
  me) [1] * e(O(O)) + C(C)(C)(=C(C(C)(C)(C))) => O(O) + e(C(C(C)(C)(C))(=C(C)(C)))
bsCH) [6] * e(C(C)(C)(C(C(C)(C)))) => C(=C)(C)(C(C(C)(C))) + e(H)
bsCH) [2] * e(C(C)(C)(C(C(C)(C)))) => C(C)(C)(=C(C(C)(C))) + e(H)
bsCC) [1] * e(C(C)(C)(C(C(C)(C)))) => C(C)(C)(=C) + e(C(C)(C))
  ox) [2] * O(=O) + e(C(C)(C)(C(C(C)(C)))) => C(=C)(C)(C(C(C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(C)(C(C(C)(C)))) => C(C)(C)(=C(C(C)(C))) + e(O(O))
bsCH) [3] * e(C(C)(C(C(C)(C)(C)))) => C(=C)(C(C(C)(C)(C))) + e(H)
bsCH) [2] * e(C(C)(C(C(C)(C)(C)))) => C(C)(=C(C(C)(C)(C))) + e(H)
bsCC) [1] * e(C(C)(C(C(C)(C)(C)))) => C(C)(=C) + e(C(C)(C)(C))
  ox) [1] * O(=O) + e(C(C)(C(C(C)(C)(C)))) => C(=C)(C(C(C)(C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(C(C(C)(C)(C)))) => C(C)(=C(C(C)(C)(C))) + e(O(O))
bsCH) [6] * e(C(C)(C)) => C(=C)(C) + e(H)
  ox) [2] * O(=O) + e(C(C)(C)) => C(=C)(C) + e(O(O))
bsCC) [3] * e(C(C(C)(C)(C))) => C(=C(C)(C)) + e(C)
bsCH) [1] * e(C(C(C)(C))) => C(=C(C)(C)) + e(H)
bsCC) [2] * e(C(C(C)(C))) => C(=C(C)) + e(C)
  ox) [1] * O(=O) + e(C(C(C)(C))) => C(=C(C)(C)) + e(O(O))
bsCH) [9] * e(C(C)(C)(C)) => C(=C)(C)(C) + e(H)
  ox) [3] * O(=O) + e(C(C)(C)(C)) => C(=C)(C)(C) + e(O(O))
```

```
bsCC) [2] * e(C(C(C)(C)(C(C(C)(C))))) => C(=C(C)(C(C(C)(C)))) + e(C)
bsCC) [1] * e(C(C(C)(C)(C(C(C)(C))))) => C(=C(C)(C)) + e(C(C(C)(C)))
bsCH) [1] * e(C(C(C)(C(C(C)(C)(C))))) => C(=C(C)(C(C(C)(C)(C)))) + e(H)
bsCC) [1] * e(C(C(C)(C(C(C)(C)(C))))) => C(=C(C(C)(C)(C)))) + e(C)
bsCC) [1] * e(C(C(C)(C(C(C)(C)(C))))) => C(=C(C)) + e(C(C(C)(C)(C)))
  ox) [1] * O(=O) + e(C(C(C)(C(C(C)(C)(C))))) => C(=C(C)(C(C(C)(C)(C)))) + e(O(O))
bsCH) [6] * e(C(C(C)(C)(C(C(C)(C)(C)))) => C(=C)(C)(C(C(C)(C)(C))) + e(H)
bsCH) [2] * e(C(C(C)(C)(C(C(C)(C)(C)))) => C(C)(C)(=C(C(C)(C)(C))) + e(H)
bsCC) [1] * e(C(C(C)(C)(C(C(C)(C)(C)))) => C(C)(C)(=C) + e(C(C)(C)(C))
  ox) [2] * O(=O) + e(C(C)(C)(C(C(C)(C)(C)))) => C(=C)(C)(C(C(C)(C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(C)(C(C(C)(C)(C)))) => C(C)(C)(=C(C(C)(C)(C))) + e(O(O))
bsCH) [1] * e(C(C(C)(C))(C(C)(C)(C))) => C(=C(C)(C))(C(C)(C)(C)) + e(H)
bsCC) [2] * e(C(C(C)(C))(C(C)(C)(C))) => C(=C(C))(C(C)(C)(C)) + e(C)
bsCC) [3] * e(C(C(C)(C))(C(C)(C)(C))) => C(C(C)(C))(=C(C)(C)) + e(C)
  ox) [1] * O(=O) + e(C(C(C)(C))(C(C)(C)(C))) => C(=C(C)(C))(C(C)(C)(C)) + e(O(O))
bsCC) [1] * e(C(=C(C)(C(C(C)(C))))) => C(#C(C(C(C)(C)))) + e(C)
bsCC) [1] * e(C(=C(C)(C(C(C)(C))))) => C(#C(C)) + e(C(C(C)(C)))
bsCC) [1] * e(C(C(=C)(C(C(C)(C))))) => C(=C(=C)) + e(C(C(C)(C)))
bsCH) [1] * e(C(C(C)(C(C(=C)(C))))) => C(=C(C)(C(C(=C)(C)))) + e(H)
bsCC) [1] * e(C(C(C)(C(C(=C)(C))))) => C(=C(C(C(=C)(C)))) + e(C)
bsCC) [1] * e(C(C(C)(C(C(=C)(C))))) => C(=C(C)) + e(C(C(=C)(C)))
  ox) [1] * O(=O) + e(C(C(C)(C(C(=C)(C))))) => C(=C(C)(C(C(=C)(C)))) + e(O(O))
bsCH) [6] * e(C(C(C)(C)(C(C(=C)(C)))) => C(=C)(C)(C(C(=C)(C))) + e(H)
bsCH) [2] * e(C(C(C)(C)(C(C(=C)(C)))) => C(C)(C)(=C(C(=C)(C))) + e(H)
bsCC) [1] * e(C(C(C)(C)(C(C(=C)(C)))) => C(C)(C)(=C) + e(C(=C)(C))
  ox) [2] * O(=O) + e(C(C)(C)(C(C(=C)(C)))) => C(=C)(C)(C(C(=C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(C)(C(C(=C)(C)))) => C(C)(C)(=C(C(=C)(C))) + e(O(O))
bsCH) [1] * e(C(C(C)(C))(C(=C)(C))) => C(=C(C)(C))(C(=C)(C)) + e(H)
bsCC) [2] * e(C(C(C)(C))(C(=C)(C))) => C(=C(C))(C(=C)(C)) + e(C)
bsCC) [1] * e(C(C(C)(C))(C(=C)(C))) => C(C(C)(C))(=C(=C)) + e(C)
  ox) [1] * O(=O) + e(C(C(C)(C))(C(=C)(C))) => C(=C(C)(C))(C(=C)(C)) + e(O(O))
bsCC) [1] * e(C(C(C)(=C(C(C)(C))))) => C(=C(=C(C(C)(C)))) + e(C)
bsCH) [1] * e(C(C(C)(C(=C(C)(C))))) => C(=C(C)(C(=C(C)(C)))) + e(H)
bsCC) [1] * e(C(C(C)(C(=C(C)(C))))) => C(=C(C(=C(C)(C)))) + e(C)
bsCC) [1] * e(C(C(C)(C(=C(C)(C))))) => C(=C(C)) + e(C(=C(C)(C)))
  ox) [1] * O(=O) + e(C(C(C)(C(=C(C)(C))))) => C(=C(C)(C(=C(C)(C)))) + e(O(O))
bsCH) [6] * e(C(C(C)(C)(C(=C(C)(C)))) => C(=C)(C)(C(=C(C)(C))) + e(H)
bsCH) [1] * e(C(C(C)(C)(C(=C(C)(C)))) => C(C)(C)(=C(=C(C)(C))) + e(H)
  ox) [2] * O(=O) + e(C(C)(C)(C(=C(C)(C)))) => C(=C)(C)(C(=C(C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(C)(C(=C(C)(C)))) => C(C)(C)(=C(=C(C)(C))) + e(O(O))
bsCH) [1] * e(C(C(C)(C))(=C(C)(C))) => C(=C(C)(C))(=C(C)(C)) + e(H)
bsCC) [2] * e(C(C(C)(C))(=C(C)(C))) => C(=C(C))(=C(C)(C)) + e(C)
bsCC) [2] * e(C(C(C)(C))(=C(C)(C))) => C(C(C)(C))(#C(C)) + e(C)
  ox) [1] * O(=O) + e(C(C(C)(C))(=C(C)(C))) => C(=C(C)(C))(=C(C)(C)) + e(O(O))
bsCC) [1] * e(C(C(C)(=C))) => C(=C(=C)) + e(C)
bsCC) [2] * e(C(=C(C)(C))) => C(#C(C)) + e(C)
bsCC) [1] * e(C(=C(C(C(C)(C)(C))))) => C(#C) + e(C(C(C)(C)(C)))
  ox) [1] * O(=O) + e(C(=C(C(C(C)(C)(C))))) => C(#C(C(C(C)(C)(C)))) + e(O(O))
bsCC) [2] * e(C(C(C)(C)(C(C(=C))))) => C(=C(C)(C(C(=C)))) + e(C)
bsCC) [1] * e(C(C(C)(C)(C(C(=C))))) => C(=C(C)(C)) + e(C(C(=C)))
bsCH) [1] * e(C(C(C)(C)(C))(C(=C))) => C(C(C)(C)(C))(=C(=C)) + e(H)
bsCC) [3] * e(C(C(C)(C)(C))(C(=C))) => C(=C(C)(C))(C(=C)) + e(C)
```

52

```
  ox) [1] * O(=O) + e(C(C(C)(C)(C))(C(=C))) => C(C(C)(C)(C))(=C(=C)) + e(O(O))
bsCH) [2] * e(C(=C)(C(C(C)(C)(C)))) => C(=C)(=C(C(C)(C)(C))) + e(H)
bsCC) [1] * e(C(=C)(C(C(C)(C)(C)))) => C(=C)(=C) + e(C(C)(C)(C))
  ox) [1] * O(=O) + e(C(=C)(C(C(C)(C)(C)))) => C(#C)(C(C(C)(C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(=C)(C(C(C)(C)(C)))) => C(=C)(=C(C(C)(C)(C))) + e(O(O))
bsCH) [1] * e(C(C(=C(C(C)(C)(C))))) => C(=C(=C(C(C)(C)(C)))) + e(H)
  ox) [1] * O(=O) + e(C(C(=C(C(C)(C)(C))))) => C(=C(=C(C(C)(C)(C)))) + e(O(O))
bsCC) [2] * e(C(C(C)(C)(C(=C(C))))) => C(=C(C)(C(=C(C)))) + e(C)
bsCC) [1] * e(C(C(C)(C)(C(=C(C))))) => C(=C(C)(C)) + e(C(=C(C)))
bsCC) [3] * e(C(C(C)(C)(C))(=C(C))) => C(=C(C)(C))(=C(C)) + e(C)
bsCC) [1] * e(C(C(C)(C)(C))(=C(C))) => C(C(C)(C)(C))(#C) + e(C)
  ox) [1] * O(=O) + e(C(C(C)(C)(C))(=C(C))) => C(C(C)(C)(C))(#C(C)) + e(O(O))
bsCH) [3] * e(C(C)(=C(C(C)(C)(C)))) => C(=C)(=C(C(C)(C)(C))) + e(H)
bsCC) [1] * e(C(C)(=C(C(C)(C)(C)))) => C(C)(#C) + e(C(C)(C)(C))
  ox) [1] * O(=O) + e(C(C)(=C(C(C)(C)(C)))) => C(=C)(=C(C(C)(C)(C))) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(=C(C(C)(C)(C)))) => C(C)(#C(C(C)(C)(C))) + e(O(O))
bsCH) [1] * e(C(C(=C))) => C(=C(=C)) + e(H)
  ox) [1] * O(=O) + e(C(C(=C))) => C(=C(=C)) + e(O(O))
bsCC) [1] * e(C(=C(C))) => C(#C) + e(C)
  ox) [1] * O(=O) + e(C(=C(C))) => C(#C(C)) + e(O(O))
bsCH) [3] * e(C(C)(=C)) => C(=C)(=C) + e(H)
  ox) [1] * O(=O) + e(C(C)(=C)) => C(=C)(=C) + e(O(O))
  ox) [1] * O(=O) + e(C(C)(=C)) => C(C)(#C) + e(O(O))
bsCC) [1] * e(C(C(C(C(C)(C)(C)))(=C))) => C(=C(=C)) + e(C(C(C)(C)(C)))
bsCC) [2] * e(C(C(C)(C)(C(C(C)(=C))))) => C(=C(C)(C(C(C)(=C)))) + e(C)
bsCC) [1] * e(C(C(C)(C)(C(C(C)(=C))))) => C(=C(C)(C)) + e(C(C(C)(=C)))
bsCC) [3] * e(C(C(C)(C)(C))(C(C)(=C))) => C(=C(C)(C))(C(C)(=C)) + e(C)
bsCC) [1] * e(C(C(C)(C)(C))(C(C)(=C))) => C(C(C)(C)(C))(=C(=C)) + e(C)
bsCC) [1] * e(C(=C(C)(C(C(C)(C)(C))))) => C(#C(C(C(C)(C)(C)))) + e(C)
bsCC) [1] * e(C(=C(C)(C(C(C)(C)(C))))) => C(#C(C)) + e(C(C(C)(C)(C)))
bsCC) [1] * e(C(C(C)(=C(C(C)(C)(C))))) => C(=C(=C(C(C)(C)(C)))) + e(C)
bsCC) [2] * e(C(C(C)(C)(C(=C(C)(C))))) => C(=C(C)(C(=C(C)(C)))) + e(C)
bsCC) [1] * e(C(C(C)(C)(C(=C(C)(C))))) => C(=C(C)(C)) + e(C(=C(C)(C)))
bsCC) [3] * e(C(C(C)(C)(C))(=C(C)(C))) => C(=C(C)(C))(=C(C)(C)) + e(C)
bsCC) [2] * e(C(C(C)(C)(C))(=C(C)(C))) => C(C(C)(C)(C))(#C(C)) + e(C)
```

Termination stage:

```
co) [1] * e(C) + e(C(C(=C))) => C(C(C(=C)))
di) [1] * e(C) + e(C(C(=C))) => C + C(=C(=C))
co) [1] * e(H) + e(C(C(=C))) => C(C(=C))
di) [1] * e(H) + e(C(C(=C))) => H(H) + C(=C(=C))
co) [1] * e(O(O)) + e(C(C(=C))) => O(O)(C(C(=C)))
di) [1] * e(O(O)) + e(C(C(=C))) => O(O) + C(=C(=C))
```

Statistics:

- Total number of reactions: 222

- Unimolecular initiations by **C–C** bond breaking [uiCC]: 4

- Unimolecular initiations by **C–H** bond breaking [uiCH]: 4

- Bimolecular initiations [bi]: 4

- Metathesis reactions [me]: 99

- Ipso [ipso]: 0

- Oxidation of free radicals [ox]: 31

- Unimolecular decomposition by beta-scission of **C–C** bond [bsCC]: 49

- Unimolecular decomposition by beta-scission of **C–C** bond [bsCH]: 25

- Combination of free radicals [co]: 3

- Disproportionation of free radicals [di]: 3

B.2 The Mechanism for Ethylcyclohexane

Input:

```
1 C(C(C(C(C1))))(C1(C(C)))
1 O(=O)
```

Initiation stage:

```
uiCC) [1] * C(C(C(C(C1))))(C1(C(C))) => e(C) + e(C(C1(C(C(C(C(C1)))))))
uiCC) [1] * C(C(C(C(C1))))(C1(C(C))) => e(C(C)) + e(C1(C(C(C(C(C1)))))))
uiCH) [4] * C(C(C(C(C1))))(C1(C(C))) => e(H) + e(C1(C(C(C(C(C1(C(C))))))))
uiCH) [4] * C(C(C(C(C1))))(C1(C(C))) => e(H) + e(C(C1)(C(C(C(C(C1(C(C)))))))
uiCH) [2] * C(C(C(C(C1))))(C1(C(C))) => e(H) + e(C(C(C1))(C(C(C1(C(C))))))
uiCH) [3] * C(C(C(C(C1))))(C1(C(C))) => e(H) + e(C(C(C1(C(C(C(C(C1)))))))))
uiCH) [2] * C(C(C(C(C1))))(C1(C(C))) => e(H) + e(C(C)(C1(C(C(C(C(C1)))))))
uiCH) [1] * C(C(C(C(C1))))(C1(C(C))) => e(H) + e(C1(C(C))(C(C(C(C(C1)))))))
  bi) [4] * O(=O) + C(C(C(C(C1))))(C1(C(C))) => e(O(O)) + e(C1(C(C(C(C(C1(C(C))))))))
  bi) [4] * O(=O) + C(C(C(C(C1))))(C1(C(C))) => e(O(O)) + e(C(C1)(C(C(C(C(C1(C(C))))))))
  bi) [2] * O(=O) + C(C(C(C(C1))))(C1(C(C))) => e(O(O)) + e(C(C(C1))(C(C(C1(C(C)))))))
  bi) [3] * O(=O) + C(C(C(C(C1))))(C1(C(C))) => e(O(O)) + e(C(C(C1(C(C(C(C(C1))))))))
  bi) [2] * O(=O) + C(C(C(C(C1))))(C1(C(C))) => e(O(O)) + e(C(C)(C1(C(C(C(C(C1))))))))
  bi) [1] * O(=O) + C(C(C(C(C1))))(C1(C(C))) => e(O(O)) + e(C1(C(C))(C(C(C(C(C1)))))))
```

Propagation stage:
Due to the very large number of reactions applied during the propagation stage we do not include them here.

Termination stage:

```
co) [1] * e(C) + e(C(C(=C))) => C(C(C(=C)))
di) [1] * e(C) + e(C(C(=C))) => C + C(=C(=C))
co) [1] * e(C) + e(C(C(=C(C)))) => C(C(C(=C(C))))
di) [1] * e(C) + e(C(C(=C(C)))) => C + C(=C(=C(C)))
co) [1] * e(C) + e(C(C(#C))) => C(C(C(#C)))
co) [1] * e(H) + e(C(C(=C))) => C(C(=C))
di) [1] * e(H) + e(C(C(=C))) => H(H) + C(=C(=C))
```

```
co) [1] * e(H) + e(C(C(=C(C)))) => C(C(=C(C)))
di) [1] * e(H) + e(C(C(=C(C)))) => H(H) + C(=C(=C(C)))
co) [1] * e(H) + e(C(C(#C))) => C(C(#C))
co) [1] * e(O(O)) + e(C(C(=C))) => O(O)(C(C(=C)))
di) [1] * e(O(O)) + e(C(C(=C))) => O(O) + C(=C(=C))
co) [1] * e(O(O)) + e(C(C(=C(C)))) => O(O)(C(C(=C(C))))
di) [1] * e(O(O)) + e(C(C(=C(C)))) => O(O) + C(=C(=C(C)))
co) [1] * e(O(O)) + e(C(C(#C))) => O(O)(C(C(#C)))
```

Statistics:

- Total number of reactions: 946

- Unimolecular initiations by **C–C** bond breaking [uiCC]: 2

- Unimolecular initiations by **C–H** bond breaking [uiCH]: 6

- Bimolecular initiations [bi]: 6

- Metathesis reactions [me]: 336

- Ipso [ipso]: 0

- Oxidation of free radicals [ox]: 221

- Unimolecular decomposition by beta-scission of **C–C** bond [bsCC]: 178

- Unimolecular decomposition by beta-scission of **C–C** bond [bsCH]: 182

- Combination of free radicals [co]: 9

- Disproportionation of free radicals [di]: 6