面向对象语言的化学语义

闫 伟 卢炎生
(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 化学计算模型是基于化学反应和计算之间比喻的并行计算模型,其内在的并行性及不确定性可以有效地消除与计算逻辑本身无关的人为顺序性,从而能够以更为直观和抽象的方式来描述并行计算。但也正是由于其内在的并行性和高层抽象性,使得特定的控制机制难以描述。高阶化学编程语言是对传统化学计算模型的扩展和泛化,在保留传统化学计算特征的同时,不仅可以有效地描述传统的控制机制,也可以方便地定义新的控制机制。通过从简单面向对象语言到高阶化学语言的转换,给出了面向对象语言的一种化学语义描述,提供了一种描述面向对象系统的新视角,也为结合面向对象编程和化学编程提供了一种可能。

关键词 化学计算,并行性,控制机制,面向对象,高阶化学语言,化学语义
中图法分类号 TP301 **文献标识码** A

Chemical Semantics of Object Oriented Languages

YAN Wei LU Yan-sheng
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract Chemical computation model is a kind of parallel computation model based on the chemical reaction metaphor. The inherent properties of chemical computation model are nondeterminism and parallelism, which can effectively get rid of artificial sequentiality not imposed by the logic of the computation and allow to describe programs in an intuitive and abstract way. However, such properties also make it rather difficult to model special control mechanisms. Higher order chemical language extends and generalizes the conventional chemical programming languages. Classical control mechanisms can be modeled in higher order chemical language, and new control mechanisms can also be defined. A simple object oriented language is transformed into high order chemical language which gives the chemical semantics of the simple object oriented language. Such chemical semantics provide a new view of object oriented systems and a possible way to combine object paradigm and chemical paradigm.

Keywords Chemical computing, Parallelism, Control mechanism, Object orientation, High-order chemical language, Chemical semantics

1 引言

化学计算模型是基于计算和化学反应之间的比喻[1],即数据空间比喻为"化学溶液",数据元素比喻为"分子",数据操作比喻为"化学反应",计算完成比喻为"反应终止"。化学反应是由反应规则描述的,反应规则包含反应条件和反应动作,当反应条件满足时,相应的分子被消耗,生成新的分子。由相同或不同的反应规则描述的反应可以同时进行,除了必须满足反应条件,额外的约束是同一个分子只能参与一个反应。因此,化学计算模型反映了可自由交互的原子值集的全局演变,其基本性质为不确定性和并行性。例如,下面是两个简单的 Gamma 程序[2]:

$\max: x, y \rightarrow x \Leftarrow x \geqslant y$

$prime: x, y \rightarrow y \Leftarrow multiple(x, y)$

式中,⇐左部为反应动作,⇐右部为反应条件。当 max 应用到包含整数的多重集 M,其中任意一对元素 x, y 如果满足反应条件 $x \geqslant y$,则被替换为元素 x。当反应终止时,M 中只包含最大元素,显然 max 是计算 M 中的最大元素。当 *prime* 应用到集合 $N=\{2,\cdots,n\}$,$n \in \mathbf{N}$,其中任意一对元素 x, y 如果满足反应条件 $multiple(x, y)$,即 x 为 y 的倍数,则被替换为元素 y。当反应终止时,N 中只包含从 2 到 n 的所有质数。Gamma 能有效地消除与算法逻辑本身无关的人为顺序性,例如,在 max 和 *prime* 中没有人为地规定元素比较的顺序,从而能够以更为直观和抽象的方式来描述并行计算,但也正是由于化学计算模型内在的不确定性和并行性,使得特定的控制机制难于描述[3,8,9]。例如,如果想计算 N 中的最大质数,仅仅把 max 和 *prime* 应用于 N 并不能保证总是得到正确的结果,因为没有方式来显式地控制规则的反应顺序,所以任意不同的满足反应条件的元素对可以以任意的顺序进行反应。高阶化学编程语言是对传统化学计算模型的扩展和泛

到稿日期:2011-04-19 返修日期:2011-07-03 本文受国家部委项目(513150601)资助。
闫 伟(1978—),男,博士生,主要研究方向为程序设计语言理论、面向方面的软件开发,E-mail:yanwei@smail.hust.edu.cn;卢炎生(1949—),男,教授,博士生导师,主要研究方向为数据库系统、软件工程、数据挖掘。

化[5,6]，其最大的不同在于多重集中的元素可以不仅仅是原子值，多重集和程序都可以是其元素，即“化学溶液”和“反应规则”都可以看作“分子”。在保留传统化学计算特征的同时，高阶化学编程语言可以有效地描述某些传统的控制机制[7]，如简单顺序控制、并发过程间的同步消息传递、基于共享变量的通信、Petri nets 等，也可以定义新的控制机制。

本文首先给出化学演算 $\gamma_{\mathcal{RM}}$，并描述其语法和语义；然后定义了一个简单的顺序面向对象语言 $\mathcal{OJ}$，并给出一种自然操作语义；接下来形式地定义了从 $\mathcal{OJ}$ 到 $\gamma_{\mathcal{RM}}$ 的转换，从而给出了面向对象语言的一种化学语义描述。最后，分析相关工作并指出了未来的研究方向。

2 $\gamma_{\mathcal{RM}}$ 演算

$\gamma_{\mathcal{RM}}$ 演算是对 γ 演算[5,6]的扩展，其语法如图 1 所示。

$$m\in Molecule \quad p\in Pattern \quad r\in RuleName$$
$$t\in\mathcal{T}(\Sigma,V) \quad x\in V \quad c\in Condition\subseteq\mathcal{T}(\Sigma,V)$$
$$m::= t\mid m_1:m_2\mid m_1,m_2\mid\langle\rangle\mid\langle m\rangle\mid !\ m\mid m\lhd p\mid p\rhd$$
$$\mid\gamma p[c].\ m\mid r=\gamma p[c].\ m\mid rec\ r=\gamma p[c].\ (m,r)$$
$$p::= x\mid\omega\mid p_1:p_2\mid p_1,p_2\mid\langle p\rangle\mid r=p\mid$$

图 1 $\gamma_{\mathcal{RM}}$ 演算的语法

一个基本的分子 t 是代数 $\mathcal{T}(\Sigma,V)$ 中的项。其中，代数 $\mathcal{T}(\Sigma,V)$ 可以是任意的，例如可以包含整数代数和布尔代数 $IntBool=\langle B,Z,+,-,*,\neg,\wedge,=,\leqslant,\cdots,-1,0,1,\cdots,true,false\rangle$，也可以是任意其它的代数。事实上，代数 $\mathcal{T}(\Sigma,V)$ 可以使用 $\gamma_{\mathcal{RM}}$ 演算来表示，为了简单起见，这里将其作为基本分子，并假设包含 $IntBool$。

通过分子构造算子可以用简单的分子来构造复杂的分子。γ 抽象 $\gamma p[m].\ m$，分子多重集 m_1,m_2，溶液 $\langle m\rangle$，分子对 $m_1:m_2$ 分别是由 γ 抽象子 $\gamma\_[\_].\ \_$，重集构造子 $\_,\_$，隔离子 $\langle\_\rangle$，结对子 $\_:\_$ 构造的。γ 抽象 $\gamma p[c].\ m$ 是反应分子，包含一个模式 p，一个反应条件 c 和生成的分子 m。γ 抽象可以命名也可以递归定义。溶液 $\langle m\rangle$ 将分子 m 与其它分子隔离开，溶液内部的分子不能参与外部的分子进行反应。

模式 p 用来匹配分子。x 可以匹配任意分子 m，$p_1:p_2$ 匹配分子对 $m_1:m_2$，p_1,p_2 匹配分子多重集，$\langle p\rangle$ 匹配惰性溶液 $\langle m\rangle$，$r=p$ 匹配命名的 γ 抽象。

分子中的自由变量和模式中出现的变量分别记为 $fv(m)$，$v(p)$，其定义如下：一个分子 m 是封闭的，如果 $fv(m)=\emptyset$，记为 $closed(m)$。

$$fv(m)=\begin{cases}\{x\}, & m=x\\ fv(m_1)\cup fv(m_2), & m=m_1,m_2\\ fv(m_1)\cup fv(m_2), & m=m_1:m_2\\ fv(m_1)\backslash v(p), & m=\gamma p[c].\ m_1\\ fv(m'), & m=\langle m'\rangle\end{cases}$$

$$v(p)=\begin{cases}\{x\}, & p=x\\ v(p_1)\cup v(p_2), & p=p_1,p_2\\ v(p_1)\cup v(p_2), & p=p_1:p_2\\ v(p'), & p=\langle p'\rangle\\ v(p'), & p=(r=p')\end{cases}$$

$\gamma_{\mathcal{RM}}$ 演算的操作语义刻画了分子的归约，包含了刻画反应的归约规则和分子等价规则。γ 抽象对应于单次反应规则，如果存在封闭分子 n 匹配模式 p，且反应条件 c 成立，则该 γ 抽象和分子 n 被消耗并生成新的分子 m。多次反应规则可以使用递归定义的 γ 抽象来实现，例如在 $rec\ r=\gamma x.\ (x,r)$ 中，由于 r 在参与反应后又重新生成，因此 r 可以多次参与反应。重复算子！应用于反应分子，可以看作该反应分子的重数为无穷，从而由其描述的反应可以并行、重复地发生。相比递归定义的 γ 抽象，重复算子！更类似于化学反应规则。$\lhd p$ 和 $p\rhd$ 可以看作特殊的反应分子，它们在其包围膜上开辟了由模式 p 刻画的特殊管道，模式 p 匹配的分子可以由该管道穿越膜。图 2 定义了 $\gamma_{\mathcal{RM}}$ 演算的归约规则。

$$\boxed{m\xrightarrow[\gamma]{}m'} \quad \mathscr{C}\in Context$$
$$\rho\in Substitution=Var\rightarrow Molecule$$
$$\mathscr{C}::=\{\cdot\}\mid\mathscr{C},m\mid m,\mathscr{C}\mid\mathscr{C}:m\mid m:\mathscr{C}\mid\langle\mathscr{C}\rangle$$
$$\frac{closed(m_2)\quad p\propto_\rho m_2\quad c[\rho]=\text{true}}{(\gamma p[c].\ m_1),m_2\xrightarrow[\gamma]{}m_1[\rho]}$$
$$\frac{(\gamma p[c].\ m_1),m_2\xrightarrow[\gamma]{}m_1[\rho]\quad \gamma=\gamma p[c].\ m_1}{\gamma,m_2\xrightarrow[\gamma]{}m_1[\rho]}$$
$$\frac{r,m\xrightarrow[\gamma]{}m'}{!\ r,m\xrightarrow[\gamma]{}m',!\ r}\qquad\frac{r,m_1\xrightarrow[\gamma]{}m'\quad r,m_1\xrightarrow[\gamma]{}m''}{!\ r,m_1,m_2\xrightarrow[\gamma]{}m',m'',!\ r}$$
$$\frac{m_1\xrightarrow[\gamma]{}m'}{m_1,m\xrightarrow[\gamma]{}m_1',m}\qquad\frac{m_1\xrightarrow[\gamma]{}m_1'\quad m_2\xrightarrow[\gamma]{}m_2'}{m_1,m_2\xrightarrow[\gamma]{}m_1',m_2'}$$
$$\frac{closed(m_1)\quad p\propto_\rho m_1}{\langle m,\lhd p\rangle,m_1\xrightarrow[\gamma]{}\langle m,m_1\rangle}\qquad\frac{closed(m_1)\quad p\propto_\rho m_1}{\langle m,m_1,p\rhd\rangle,m_1\xrightarrow[\gamma]{}\langle m\rangle,m_1}$$
$$\frac{m_1'\rightleftharpoons m_1\quad m_1\xrightarrow[\gamma]{}m_2\quad m_2\rightleftharpoons m_2'}{m_1'\xrightarrow[\gamma]{}m_2'}\qquad\frac{m\xrightarrow[\gamma]{}m'}{\mathscr{C}\{m\}\xrightarrow[\gamma]{}\mathscr{C}\{m'\}}$$

图 2 $\gamma_{\mathcal{RM}}$ 演算的归约规则

替换 $\rho\in Substitution=Var\rightarrow Molecule$ 是一个从 Var 到 $Molecule$ 的部分函数。谓词 $p\propto_\rho m$ 表示在替换 ρ 下模式 p 匹配分子 m，即 $p[\rho]\rightleftharpoons m$。谓词 $p\propto_\rho m$ 定义如下：

$$\boxed{p\propto_\rho m}\qquad\frac{}{x\propto_{\{x\mapsto m\}}m}\qquad\frac{p\propto_\rho m}{\langle p\rangle\propto_\rho\langle m\rangle}\qquad\frac{p\propto_\rho m}{r=p\propto_\rho r=m}$$
$$\frac{p_1\propto_{\rho_1}m_1\quad p_2\propto_{\rho_2}m_2\quad \rho_1\doteq\rho_2}{p_1,p_2\propto_{\rho_1\cup\rho_2}m_1,m_2}$$
$$\frac{p_1\propto_{\rho_1}m_1\quad p_2\propto_{\rho_2}m_2\quad \rho_1\doteq\rho_2}{p_1:p_2\propto_{\rho_1\cup\rho_2}m_1:m_2}$$
$$\frac{p'\rightleftharpoons p\quad p\propto_\rho m\quad m\rightleftharpoons m'}{p'\propto_\rho m'}$$
$$\rho_1\doteq\rho_2\Leftrightarrow\forall x\in dom(\rho_1)\cap dom(\rho_2)\cdot\rho_1(x)=\rho_2(x)$$

$\rho_1\doteq\rho_2$ 表示替换 ρ_1 和 ρ_2 是兼容的，即在其共同的定义域上 ρ_1 和 ρ_2 具有相同的像。替换代入记为 $m[\rho]$，定义为：

$$m[\rho]=\begin{cases}\rho(x), & m=x\wedge x\in dom(\rho)\\ \rho(m_1),\rho(m_2), & m=m_1,m_2\\ \rho(m_1):\rho(m_2), & m=m_1:m_2\\ \langle\rho(m')\rangle, & m=\langle m'\rangle\\ \gamma p[c].\ m_1[\rho\upharpoonright v(p)], & m=\gamma p[c].\ m_1\end{cases}$$

$\rightleftharpoons\subseteq Molecule\times Molecule$ 为分子上的等价关系，定义为：

$$\boxed{m\rightleftharpoons m'}\qquad\frac{}{m\rightleftharpoons m}\qquad\frac{m_1\rightleftharpoons m_2}{m_2\rightleftharpoons m_1}\qquad\frac{m_1\rightleftharpoons m_2\quad m_2\rightleftharpoons m_3}{m_1\rightleftharpoons m_3}$$
$$\frac{}{m_1,m_2\rightleftharpoons m_2,m_1}\qquad\frac{}{!\ r\rightleftharpoons r,!\ r}$$
$$\frac{}{m_1,(m_2,m_3)\rightleftharpoons(m_1,m_2),m_3}\qquad\frac{m_1\rightleftharpoons m_1'\quad m_2\rightleftharpoons m_2'}{m_1:m_2\rightleftharpoons m_1':m_2'}$$
$$\frac{m\rightleftharpoons m'}{\langle m\rangle\rightleftharpoons\langle m'\rangle}\qquad\frac{}{\langle\langle m\rangle\rangle\rightleftharpoons\langle m\rangle}\qquad\frac{m\rightleftharpoons m'}{\gamma p[c].\ m\rightleftharpoons\gamma p[c].\ m'}$$
$$\frac{p'=p[\rho]\quad dom(\rho)=v(p)\quad ran(\rho)\subseteq Var\quad ran(\rho)\cap fv(m)=\emptyset}{\gamma p[c].\ m\rightleftharpoons\gamma p'[c[\rho]].\ m[\rho]}$$

3 面向对象语言 $\mathcal{SJ}$

$\mathcal{SJ}$ 是一个类似于 Java[11] 的简单面向对象语言，可以看作是对 Featherwight Java[12] 的命令式扩展。为了简单起见，$\mathcal{SJ}$ 只包含了面向对象语言的基本特征。一个 $\mathcal{SJ}$ 程序包含一组类定义 $Cdec$ 和一个语句 S。一个类的定义 class C extends $D\{Fdec\ \kappa\ Mdec\}$ 包含类名 C、父类 D、成员域 $Fdec$、构造函数 κ 和成员方法 $Mdec$。一个方法定义 $C\ m(D_1\ x_1,\cdots,D_m\ x_m)\{S;\text{return } e\}$ 包含方法名 m、返回类型 C、参数 $D_1\ x_1,\cdots,D_m\ x_m$ 和方法体 S；return e。表达式 e 可以是常量 c，变量 x，对象成员域访问 $e.f$，当前对象引用 this 和常量 nil。语句 S 可以是空语句 skip，变量声明 $T\ x$，变量赋值语句 $x:=e$，对象成员域赋值 $e.f:=e$，对象方法调用 $e.m(\vec{e})$，对象创建 $x:=\text{new } C(\vec{e})$，条件语句 if e then $\{S_1\}$ else $\{S_2\}$，循环语句 while e do $\{S\}$，返回语句 return e，语句的顺序合成 $S_1;S_2$。

$\mathcal{SJ}$ 是一个类型化的语言，其静态语义包含一个类型系统，定义了类型良好的程序应满足的条件。$\mathcal{SJ}$ 的动态语义定义了类型良好的程序的执行。$\mathcal{SJ}$ 的类型系统暂时不是本文所考虑的问题，所以只给出 $\mathcal{SJ}$ 的动态语义，并总是假设程序是类型良好的。$\mathcal{SJ}$ 的动态语义定义了一个变迁系统，$TS_{\mathcal{SJ}}=(Con;\rightsquigarrow,\rightarrow\subseteq Con\times Con)$，其中 Con 为配置集，$\rightsquigarrow$，$\rightarrow$ 为变迁关系。一个配置 $con\in Con$ 可以具有形式 $\langle CT,S,s,\sigma\rangle$，$\langle CT,e,s,\sigma\rangle$，$\langle CT,v,s,\sigma\rangle$ 或者 $\langle CT,s,\sigma\rangle$。一个变迁 $\langle CT,S,s,\sigma\rangle\rightsquigarrow\langle CT,s',\sigma'\rangle$ 表示语句 S 在类环境 CT、变量环境 s 和状态 σ 下执行得到变量环境 s' 和状态 σ'，变迁 $\langle CT,e,s,\sigma\rangle\rightsquigarrow\langle CT,v,s',\sigma'\rangle$ 表示表达式 e 在类环境 CT、变量环境 s 和状态 σ 下求值得到 v、变量环境 s' 和状态 σ'。由于 $\mathcal{SJ}$ 不包含动态定义的类，类环境在执行过程中保持不变，并且表达式求值没有副效应，求值前后的变量环境 s 和状态 σ 保持不变，因此语句变迁可以记为 $CT\vdash\langle S,s,\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle$，表达式变迁可记为 $CT,s,\sigma\vdash e\rightarrow v$。$\mathcal{SJ}$ 的抽象语法和动态语义如图 3 所示，其中辅助函数 $mbody(C,m)$ 和 $cbody(C)$ 分别返回类 C 中方法 m 及构造函数的定义。

$aop\in\{+,-,*\}\quad bop\in\{not,and,or\}\quad rop\in\{=,<,>,\leq,\geq\}\quad x\in Var\quad e\in Exp\quad S\in Stmt\quad P\in Prog$
$c\in Constant=\{\ldots,-1,0,1,\ldots,true,false,null\}\quad T\in Type=CName\cup\{int,bool,void\}\quad C,D\in CName\quad Cdec\in CDec$
$f\in FName\quad Fdec\in FDec\quad m\in MName\quad Mdec\in MDec\quad \kappa\in Constr\quad disjoint(Var,CName,FName,MName,Constant)$

$Cdec::=class\ C\ extends\ D\ \{Fdec\ \kappa\ Mdec\}\ Cdec\mid\varepsilon\quad Fdec::=C\ f;Fdec\mid\varepsilon$
$Mdec::=T\ m(D_1\ x_1,\ldots,D_m\ x_m)\ \{S;return\ e\}\ Mdec\mid\varepsilon\quad \kappa::=C(D_1\ x_1,\ldots,D_n\ x_n)\ \{super(\vec{e});S\}$
$e::=c\mid x\mid e.f\mid this\mid aop(\vec{e})\mid bop(\vec{e})\mid rop(\vec{e})\quad P::=Cdec\ \{S\}$
$S::=skip\mid T\ x\mid x:=e\mid x:=new\ C(\vec{e})\mid x:=e.m(\vec{e})\mid e.f:=e\mid e.m(\vec{e})\mid return\ e\mid S_1;S_2\mid if\ e\ then\ \{S_1\}\ else\ \{S_2\}\mid while\ e\ do\ \{S\}$

$Object=CName\times Fields\quad v\in Val=(\mathbb{O}\times CName)\cup\{null\}\quad F\in Fields=FName\rightharpoonup Val\quad \sigma\in State=\mathbb{O}\rightharpoonup Object\quad s\in Stack=Var\rightharpoonup Val$

$\boxed{CT,s,\sigma\vdash e\rightarrow v}\quad (Var)\dfrac{s(x)=v}{CT,s,\sigma\vdash x\rightarrow v}\quad (This)\dfrac{s(this)=v}{CT,s,\sigma\vdash this\rightarrow v}\quad (Const)\dfrac{}{CT,s,\sigma\vdash c\rightarrow V_c}$

$(FieldAcc)\dfrac{CT,s,\sigma\vdash e\rightarrow v'\quad \sigma(v')=(C,F)\quad F(f)=v}{CT,s,\sigma\vdash e.f\rightarrow v}\quad (Op)\dfrac{CT,s,\sigma\vdash e_1\rightarrow v_1\quad\cdots\quad CT,s,\sigma\vdash e_n\rightarrow v_n\quad op(v_1,\ldots v_n)=v}{CT,s,\sigma\vdash op(e_1,\ldots,e_n)\rightarrow v}$

$\boxed{CT\vdash mbody(C,m)=(\vec{x},S)}\quad \dfrac{class\ C\ extends\ D\ \{\ \ldots\ T\ m(D_1\ x_1,\ldots,D_m\ x_m)\{S\}\ \ldots\ \}\in CT}{CT\vdash mbody(C,m)=(x_1,\ldots,x_m,S)}$

$\dfrac{class\ C\ extends\ D\ \{\ \ldots\ T_i\ m_i(D_1\ x_1,\ldots,D_m\ x_m)\{S\}\ \ldots\ ^{i\in 1..n}\}\in CT\quad \forall i\in 1..n\bullet m\neq m_i}{CT\vdash mbody(C,m)=mbody(D,m)}$

$\boxed{CT\vdash cbody(C)=(\vec{x},S)}\quad \dfrac{class\ C\ extends\ D\ \{\ \ldots\ C(D_1\ x_1,\ldots,D_m\ x_m)\{S\}\ \ldots\ \}\in CT}{CT\vdash cbody(C)=(x_1,\ldots,x_m,S)}$

$\boxed{CT\vdash\langle S,s,\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle}\quad (Skip)\dfrac{}{CT\vdash\langle skip,s,\sigma\rangle\rightsquigarrow\langle s,\sigma\rangle}\quad (VarDec)\dfrac{v=init(T)}{CT\vdash\langle T\ x,s,\sigma\rangle\rightsquigarrow\langle s[x\mapsto v],\sigma\rangle}\quad (VarAss)\dfrac{x\in dom(s)\quad CT,s,\sigma\vdash e\rightarrow v}{CT\vdash\langle x:=e,s,\sigma\rangle\rightsquigarrow\langle s[x\mapsto v],\sigma\rangle}$

$(FieldAss)\dfrac{CT,s,\sigma\vdash e_1\rightarrow v_1\quad CT,s,\sigma\vdash e_2\rightarrow v_2\quad \sigma(v_1)=(C,F)\quad \sigma'=\sigma[v_1\mapsto(C,F[f\mapsto v_2])]}{CT\vdash\langle e_1.f:=e_2,s,\sigma\rangle\rightsquigarrow\langle s,\sigma'\rangle}$

$(NewObj)\dfrac{\begin{array}{l}x\in dom(s)\quad CT,s,\sigma\vdash e_1\rightarrow v_1\cdots CT,s,\sigma\vdash e_n\rightarrow v_n\quad cbody(CT,C)=(x_1,\ldots,x_n,S)\\ CT\vdash\langle S,s[x_i\mapsto v_i,this\mapsto c],\sigma[c\mapsto(C,F_{null})]\rangle\rightsquigarrow\langle c,s',\sigma'\rangle\quad c\notin dom(\sigma)\end{array}}{CT\vdash\langle x:=new\ C(e_1,\ldots,e_n),s,\sigma\rangle\rightsquigarrow\langle s[x\mapsto c],\sigma'\rangle}$

$(MethInv1)\dfrac{\begin{array}{l}CT,s,\sigma\vdash e_0\rightarrow v_0\quad CT,s,\sigma\vdash e_1\rightarrow v_1\cdots CT,s,\sigma\vdash e_n\rightarrow v_n\\ \sigma(v_0)=(C,F)\quad mbody(CT,C,m)=(x_1,\ldots,x_n,S)\quad CT\vdash\langle S,s[x_i\mapsto v_i,this\mapsto v_0],\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle\end{array}}{CT\vdash\langle e_0.m(e_1,\ldots,e_n),s,\sigma\rangle\rightsquigarrow\langle s,\sigma'\rangle}$

$(MethInv2)\dfrac{\begin{array}{l}CT,s,\sigma\vdash e_0\rightarrow v_0\quad CT,s,\sigma\vdash e_1\rightarrow v_1\cdots CT,s,\sigma\vdash e_n\rightarrow v_n\\ \sigma(v_0)=(C,F)\quad mbody(CT,C,m)=(x_1,\ldots,x_n,S;return\ e)\\ CT\vdash\langle S,s[x_i\mapsto v_i,this\mapsto v_0],\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle\quad CT,s',\sigma'\vdash e\rightarrow v\end{array}}{CT\vdash\langle x:=e_0.m(e_1,\ldots,e_n),s,\sigma\rangle\rightsquigarrow\langle s[x\mapsto v],\sigma'\rangle}\quad (Seq)\dfrac{CT\vdash\langle S_1,s,\sigma\rangle\rightsquigarrow\langle s'',\sigma''\rangle\quad CT\vdash\langle S_2,s'',\sigma''\rangle\rightsquigarrow\langle s',\sigma'\rangle}{CT\vdash\langle S_1;S_2,s,\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle}$

$(IfTrue)\dfrac{CT,s,\sigma\vdash e\rightarrow T\quad CT\vdash\langle S_1,s,\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle}{CT\vdash\langle if\ e\ then\ \{S_1\}\ else\ \{S_2\},s,\sigma\rangle\rightsquigarrow\langle s,\sigma'\rangle}\quad (IfFalse)\dfrac{CT,s,\sigma\vdash e\rightarrow F\quad CT\vdash\langle S_2,s,\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle}{CT\vdash\langle if\ e\ then\ \{S_1\}\ else\ \{S_2\},s,\sigma\rangle\rightsquigarrow\langle s,\sigma'\rangle}$

$(WhileTrue)\dfrac{CT,s,\sigma\vdash e\rightarrow T\quad CT\vdash\langle S_1;while\ e\ do\ \{S_1\},s,\sigma\rangle\rightsquigarrow\langle s',\sigma'\rangle}{CT\vdash\langle while\ e\ do\ \{S_1\},s,\sigma\rangle\rightsquigarrow\langle s,\sigma'\rangle}\quad (WhileFalse)\dfrac{CT,s,\sigma\vdash e\rightarrow F}{CT\vdash\langle while\ e\ do\ \{S_1\},s,\sigma,s,\sigma\rangle\rightsquigarrow\langle s,\sigma\rangle}$

图 3 $\mathcal{SJ}$ 的抽象语法和动态语义

4 $\mathcal{SJ}$ 的 $\gamma_{\mathcal{RA}}$ 演算表示

给定一个 $\mathcal{SJ}$ 程序：

$P\equiv$class C extends $D\{\cdots\}\cdots S_P$，其对应的 $\gamma_{\mathcal{RA}}$ 演算具有形式 $\langle Class_{env},P:\langle pc:a,code,local\rangle,Object,\mathcal{R}\rangle$。其中 $Class_{env}$ 为类环境，提供了对象实例化和对象方法调用的参照；$pc:a$ 指示下一条将要执行的指令；$code$ 为指令重集，其元素 $a:\langle r\rangle$ 是一个指令地址——反应规则对或者特殊的指令 $a:instruction$；$local$ 为局部变量环境，其元素 $x:v$ 是一个变量——值对；$Object$ 为对象重集，包含当前存在的对象实

例。对象创建完成后一直处于稳定状态且具有形式 $c:C:\langle state\rangle$，当有消息到达时，对象被激活并开始参与反应直到重新到达稳定状态。$\mathcal{R}$ 为控制程序执行的反应规则集合。例如，图 4 是一个简单的 $\mathcal{SJ}$ 程序和其对应的 $\gamma_{\mathcal{RM}}$ 演算程序，以及相应的部分推导，其中推导 $\mathcal{D}_1$，$\mathcal{D}_2$，$\mathcal{D}_3$，$\mathcal{D}_4$ 类似，故省略。

```
P ≡
class Point extends Object {
    int px;
    int py;
    Point(int ix, int iy){
        super();
        this.px := ix;
        this.py := iy
    }
    void setX(int nx){
        this.px := nx
    }
    void setY(int ny){
        this.py := ny
    }
    void move(int nx, int ny){
        this.setX(nx);
        this.setY(ny)
    }
}

Point p;
p := new Point(5,10);
p.setX(10);
p.setY(5);
p.move(0,0)
```

$$\left\langle Class_{env},\ P:\left\langle pc{:}1,\ code\equiv\begin{cases}1{:}\langle declare(Point,p)\rangle\\ 2{:}new{:}Point{:}5{:}10\\ 3{:}\langle assign(p,new{:}Point)\rangle\\ 4{:}meth{:}setX{:}p{:}10\\ 5{:}meth{:}setY{:}p{:}5\\ 6{:}meth{:}move{:}p{:}0{:}0\end{cases}\right\rangle,\ run,\ \mathcal{I}',\ \mathcal{O}\right\rangle$$

$$Class_{env}\equiv\left\langle F_{Point}=px{:}int,py{:}int,\ \kappa_{Point}=\left\langle pc{:}1,code\equiv\begin{cases}1{:}\langle cval(this.px),cval(ix)\rangle\\ 2{:}\langle assign(this.px,ix)\rangle\\ 3{:}\langle eval(this.py),eval(iy)\rangle\\ 4{:}\langle assign(this.py,iy)\rangle\\ 5{:}return{:}this\end{cases}\right\rangle\right\rangle$$

$$Point{:}Object{:}\left\langle M_{Point}\equiv setX=\left\langle pc{:}1,code\equiv\begin{cases}1{:}\langle cval(this.px),cval(nx)\rangle\\ 2{:}\langle assign(this.px,nx)\rangle\\ 3{:}return\end{cases}\right\rangle\right.$$

$$\left.setY=\left\langle pc{:}1,code\equiv\begin{cases}1{:}\langle cval(this.py),cval(ny)\rangle\\ 2{:}\langle assign(this.py,ny)\rangle\\ 3{:}return\end{cases}\right\rangle\right.$$

$$\left.move=\left\langle pc{:}1,code\equiv\begin{cases}1{:}meth{:}setX{:}this{:}nx\\ 2{:}meth{:}setX{:}this{:}nx\\ 3{:}return\end{cases}\right\rangle\right\rangle$$

$$\mathcal{D}:\ \dfrac{\dfrac{\dfrac{\mathcal{D}_1}{CT\vdash\langle Point\ p;p:=new\ Point(5,10),s_0,\sigma_0\rangle\rightsquigarrow\langle s_2,\sigma_2\rangle}\quad\dfrac{\mathcal{D}_2}{CT\vdash\langle p.setX(10),s_2,\sigma_2\rangle\rightsquigarrow\langle s_3,\sigma_3\rangle}}{CT\vdash\langle Point\ p;p:=new\ Point(5,10);p.setX(10),s_0,\sigma_0\rangle\rightsquigarrow\langle s_3,\sigma_3\rangle}\quad\dfrac{\mathcal{D}_3}{CT\vdash\langle p.setY(5),s_3,\sigma_3\rangle\rightsquigarrow\langle s_4,\sigma_4\rangle}}{CT\vdash\langle Point\ p;p:=new\ Point(5,10);p.setX(10),s_0,\sigma_0\rangle\rightsquigarrow\langle s_4,\sigma_4\rangle}$$

$$\dfrac{\mathcal{D}\qquad\dfrac{\mathcal{D}_4}{CT\vdash\langle p.move(0,0),s_4,\sigma_4\rangle\rightsquigarrow\langle s_5,\sigma_5\rangle}}{CT\vdash\langle Point\ p;p:=new\ Point(5,10);p.setX(10);p.move(0,0),s_0,\sigma_0\rangle\rightsquigarrow\langle s_5,\sigma_5\rangle}$$

$$s_0=\sigma_0=\varnothing\quad s_1=\{p\mapsto null\},\sigma_1=\sigma_0\quad s_2=\{p\mapsto v\},\sigma_2=\{v\mapsto(Point,\{px\mapsto 5,py\mapsto 10\})\}\quad s_2=s_3=s_4=s_5$$
$$\sigma_4=\{v\mapsto(Point,\{px\mapsto 10,py\mapsto 10\})\}\quad\sigma_4=\{v\mapsto(Point,\{px\mapsto 10,py\mapsto 5\})\}\quad\sigma_5=\{v\mapsto(Point,\{px\mapsto 0,py\mapsto 0\})\}$$

$$\langle Class_{env},P:\langle pc{:}1,code\equiv\{1{:}\langle declare(Point,p)\rangle,2{:}new{:}Point{:}5{:}10,3{:}\langle assign(p,new{:}Point)\rangle,$$
$$4{:}meth{:}setX{:}p{:}10,5{:}meth{:}setY{:}p{:}5,6{:}meth{:}move{:}p{:}0{:}0\},run,\mathcal{I}',\mathcal{O}\rangle\ \overset{*}{\underset{\gamma}{\longrightarrow}}$$
$$\langle Class_{env},P:\langle\boxed{pc{:}2},code\equiv\{1{:}\langle declare(Point,p)\rangle,2{:}new{:}Point{:}5{:}10,3{:}\langle assign(p,new{:}Point)\rangle,$$
$$4{:}meth{:}setX{:}p{:}10,5{:}meth{:}setY{:}p{:}5,6{:}meth{:}move{:}p{:}0{:}0\},\boxed{p{:}null}\rangle,run,\mathcal{I}',\mathcal{O}\rangle\ \overset{createObj}{\underset{\gamma}{\longrightarrow}}\ \overset{*}{\underset{\gamma}{\longrightarrow}}$$
$$\langle Class_{env},P:\langle pc{:}2,code\equiv\{1{:}\langle declare(Point,p)\rangle,\boxed{2{:}new{:}Point{:}5{:}10},3{:}\langle assign(p,new{:}Point)\rangle,$$
$$4{:}meth{:}setX{:}p{:}10,5{:}meth{:}setY{:}p{:}5,6{:}meth{:}move{:}p{:}0{:}0\},p{:}null\rangle,\boxed{v{:}Point{:}\kappa_{Point}},run,\mathcal{I}',\mathcal{O}\rangle\ \overset{*}{\underset{\gamma}{\longrightarrow}}$$
$$\langle Class_{env},P:\langle\boxed{pc{:}4},code\equiv\{1{:}\langle declare(Point,p)\rangle,2{:}new{:}Point{:}5{:}10,3{:}\langle assign(p,new{:}Point)\rangle,$$
$$4{:}meth{:}setX{:}p{:}10,5{:}meth{:}setY{:}p{:}5,6{:}meth{:}move{:}p{:}0{:}0\},p{:}v\rangle,\boxed{v{:}Point{:}\langle this{:}v,px{:}5,py{:}10\rangle},run,\mathcal{I}',\mathcal{O}\rangle\ \overset{*}{\underset{\gamma}{\longrightarrow}}$$
$$\langle Class_{env},P:\langle\boxed{pc{:}7},code\equiv\{1{:}\langle declare(Point,p)\rangle,2{:}new{:}Point{:}5{:}10,3{:}\langle assign(p,new{:}Point)\rangle,$$
$$4{:}meth{:}setX{:}p{:}10,5{:}meth{:}setY{:}p{:}5,6{:}meth{:}move{:}p{:}0{:}0\},p{:}v\rangle,\boxed{v{:}Point{:}\langle this{:}v,px{:}0,py{:}0\rangle},run,\mathcal{I}',\mathcal{O}\rangle$$

图 4　$\mathcal{SJ}$ 程序及其对应的 $\gamma_{\mathcal{RM}}$ 演算程序

规则 *run* 反应后，当前指令地址对应的反应规则被提取出来，从而可以开始反应并改变程序变量的值。当前指令地址的特殊指令触发相应的反应规则，例如，$\xrightarrow{createObj}$ 表示规则 *createObj* 与 $2{:}new{:}Point{:}5{:}10$ 反应生成一个新的 *Point* 对象实例。根据 $\gamma_{\mathcal{RM}}$ 演算的性质，只有当 P 对应的溶液稳定后规则 *run* 才能继续反应。最终，整个溶液达到稳定状态，程序执行终止。接下来给出从 $\mathcal{SJ}$ 到 $\gamma_{\mathcal{RM}}$ 演算转换的具体定义。

4.1　类转换

每个类 class C extends $D\{\mathrm{Fdec}\ \kappa\ \mathrm{Mdec}\}$ 可以表示为一个分子：

$$C{:}D{:}\langle F_C\equiv\begin{cases}f_1{:}C_1,\\ \vdots\\ f_m{:}C_m\end{cases},F_I,\kappa_C{:}\langle b_\kappa\rangle,M_C\equiv\begin{cases}m_1{:}\langle b_{m_1}\rangle,\\ \vdots\\ m_n{:}\langle b_{m_n}\rangle\end{cases},M_I\rangle$$

其中包含类名 C 及其父类名 D；C 中声明的成员变量

$field_C$ 和从父类继承的成员变量 F_I；构造函数 $\kappa_C:\langle b_\kappa\rangle$；$C$ 中声明的成员方法 M_C 和从父类继承的成员方法 M_I。每个成员变量包含变量名 f_i 及其类型 C_i 和初始值 null，每个成员方法包含方法名 m 和方法体 b_m，从父类继承的方法只包含定义该方法的类名和方法名，例如 $D:m$ 表示方法 m 是在类 D 中定义的。类转换函数 $\mathscr{C}[\![\_]\!]$ 映射 $\mathscr{JJ}$ 类定义到分子上的函数。在 $\mathscr{JJ}$ 中类 Object 是所有类的父类，不包含成员变量和成员方法，定义为：

$\mathscr{C}[\![\text{Object}]\!]\langle\omega\rangle=\langle\text{Object}:\langle\rangle,\omega\rangle$

没有声明继承的类总是继承类 Object，定义为：

$$\begin{aligned}&\mathscr{C}[\![\text{class } C\{C_1\ f_1;\cdots;C_m\ f_m;C(\vec{D}\ \vec{x})\{S_C\}\\&\quad m_1(\vec{D}\ \vec{x})\{S_{m_1}\}\cdots m_n(\vec{D}\ \vec{x})\{S_{m_n}\}\}]\!]\langle\text{Object}:\langle\rangle,\omega\rangle\\&=\langle C:\text{Object}:\langle f:C,\cdots,f_m:C_m,\kappa_C:\mathscr{S}[\![S_C]\!]InitSolution(\vec{x}:\vec{D}),m_1:\mathscr{S}[\![S_{m_1}]\!]InitSolution(\vec{x}:\vec{D}),\cdots,m_n:\mathscr{S}[\![S_{m_n}]\!]InitSolution(\vec{x}:\vec{D}),\text{Object}:\langle\rangle,\omega\rangle\end{aligned}$$

显式声明继承的类，需要标记从父类继承的成员变量和方法，定义为：

$$\begin{aligned}&\mathscr{C}[\![\text{class } C \text{ extends } D\{C_1\ f_1;\cdots;C_m\ f_m;C(\vec{D}\ \vec{x})\{S_C\}\\&\quad m_1(\vec{D}\ \vec{x})\{S_{m_1}\}\cdots m_n(\vec{D}\ \vec{x})\{S_{m_n}\}\}]\!]\\&\langle D:E:\langle F_D,F_I,\cdots,M_D,M_I\rangle,\omega\rangle\\&\quad=\langle C:D:\langle f:C,\cdots,f_m:C_m,,F_D,\\&\qquad\kappa_C:\mathscr{S}[\![S_C]\!]InitSolution(\vec{x}:\vec{D}),\\&\qquad m_1:\mathscr{S}[\![S_{m_1}]\!]InitSolution(\vec{x}:\vec{D}),\\&\qquad\cdots,m_n:\mathscr{S}[\![S_{m_n}]\!]InitSolution(\vec{x}:\vec{D}),\\&\qquad D:(M_D\backslash\{m_1,\cdots,m_n\}),M_I\backslash\{m_1,\cdots,m_n\}\rangle,\\&\qquad D:E:\langle F_D,F_I,\cdots,M_D,M_I\rangle,\omega\rangle\end{aligned}$$

其中，$InitSolution(\vec{x}:\vec{D})=\langle pc:1,a:1,\vec{x}:\vec{D}\rangle$ 构建初始溶液。

类定义序列的转换为其顺序合成：

$\mathscr{C}[\![\text{class } C \text{ extends } D\{\cdots\}\ Cdec]\!]=\mathscr{C}[\![Cdec]\!]\circ\mathscr{C}[\![\text{class } C \text{ extends } D\{\cdots\}]\!]$

图 5 给出了一些简单的类定义及其 $\gamma_{\mathscr{RA}}$ 演算表示。

$$\begin{aligned}&class\ C\ \{C_1\ f;\ C()\{\ldots\}\ m_1()\{\ldots\}\}\xrightarrow{c_1}C:\{f:C_1,C:\{\ldots\},\ m_1:\{\ldots\}\rangle\\&class\ D\ extends\ C\ \{D_1\ g;\ D()\{\ldots\}\ m_2()\{\ldots\}\}\xrightarrow{c_1}\begin{cases}D:C:\{f:C_1,g:D_1,\\\quad D:\{\ldots\},m_2:\{\ldots\},C:m_1\}\end{cases}\\&class\ E\ extends\ D\ \{E_1\ h;\ E()\{\ldots\}\ m_1()\{\ldots\}\ m_3()\{\ldots\}\}\xrightarrow{c_1}\begin{cases}E:D:\{h:E_1,f:C_1,g:D_1,\\\quad E:\{\ldots\},m_1:\{\ldots\},m_3:\{\ldots\},D:m_2\}\end{cases}\end{aligned}$$

图 5 $\mathscr{JJ}$ 类转换

4.2 表达式转换

表达式转换函数映射 $\mathscr{JJ}$ 表达式到分子上的函数。由于中表达式的求值没有副效应，即不会改变当前状态，因此其子表达式的求值可以并行进行。

常量 c 转换为规则 $\gamma x.\ x,c:c$，其反应生成新的分子 $c:c$。变量 x 和 this 转换为空规则 $\gamma x.\ x$，其反应不产生任何效果。对象成员域访问 $e.\ f$ 的转换包含以下规则：$eval(e)$ 为表达式 e 对应的求值规则；规则 $\lhd v:C:\langle f:v',\omega\rangle$ 和规则 $\gamma(e:v,v:C:\langle f:v',\omega\rangle).\ e.\ f:v',v:C:\langle f:v',\omega\rangle\rhd$ 实现了一个对象成员域访问。尽管这些规则之间没有任何顺序，但是由于规则 $\gamma(e:v,v:C:\langle f:v',\omega\rangle).\ e.\ f:v',v:C:\langle f:v',\omega\rangle\rhd$ 只有当 e 求值完成且对象 $v:C:\langle f:v',\omega\rangle$ 存在时才能反应，从而保证了对象成员域访问的正确性。基本运算表达式 $op(e_1,\cdots,e_n)$ 转换为规则 $eval(e_1),\cdots,eval(e_n)$，分别对应表达式 $e_1,\cdots,e_n$ 的求值规则和 $\gamma(e_1:v_1,\cdots,e_n:v_2).\ op(e_1,\cdots,e_n):op(v_1,\cdots,v_n)$，其子表达式对应的规则可以并行反应。当子表达式求值完成，规则 $\gamma(e_1:v_1,\cdots,e_n:v_2).\ op(e_1,\cdots,e_n):op(v_1,\cdots,v_n)$ 反应得到预期的结果。表达式转换函数 $\xi[\![\_]\!]:Exp^*\to(Molecule\rightharpoonup Molecule)$ 具体定义如下：

$\xi[\![e]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(e)\rangle,\omega\rangle\quad eval(c)\equiv\gamma x.\ x,c:c$

$\xi[\![c]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(c)\rangle,\omega\rangle\quad eval(x)\equiv skip$

$\xi[\![x]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(x)\rangle,\omega\rangle\quad eval(this)\equiv skip$

$\xi[\![this]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(this)\rangle,\omega\rangle\quad eval(e.\ f)\equiv eval(e),\lhd v:C:\langle f:v',\omega\rangle,\gamma(e:v,v:C:\langle f:v',\omega\rangle).\ e.\ f:v',v:C:\langle f:v',\omega\rangle\rhd$

$\xi[\![e.\ f]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(e.\ f)\rangle,\omega\rangle\quad eval(op(e_1,\cdots,e_n))\equiv eval(e_1),\cdots,eval(e_n),\gamma(e_1:v_1,\cdots,e_n:v_2).\ op(e_1,\cdots,e_n):op(v_1,\cdots,v_n)$

$\xi[\![op(e_1,\cdots,e_n)]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(op(e_1,\cdots,e_n))\rangle,\omega\rangle$

$eval(e_1,\cdots,e_n)\equiv eval(e_1),\cdots,eval(e_n)$

$\xi[\![e_1,\cdots,e_n]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle eval(e_1,\cdots,e_n)\rangle,\omega\rangle\quad skip\equiv\gamma x.\ x,op\in aop\cup bop\cup rop,n\geqslant 1$

4.3 语句转换

语句转换函数映射 $\mathscr{JJ}$ 语句到分子上的函数。语句 $skip$ 转换为空规则 $\gamma x.\ x$。变量声明 $T\ x$ 转换为规则 $declare(T,x)$，其反应产生代表变量 x 和其初始值 init(T) 的分子 x:init(T)。变量赋值语句 $x:=e$ 的转换为表达式 e 的转换 $\xi[\![e]\!]$ 和 $Assign(x,e)$ 的顺序合成。对象实例化 $x:=$new $C(\vec{e})$ 的转换为表达式 $\vec{e}$ 的转换 $\xi[\![\vec{e}]\!]$ 和 $Assign(x,\text{new}:C)\circ NewObject(C,\vec{e})$ 的顺序合成，其中 $NewObject(C,\vec{e})$ 产生对应于对象创建的指令 new:$C:\vec{e}$，规则 $createObj$ 与之反应并生成新的对象实例。对象方法调用与对象实例化类似，不同的是 $Method(m,e,\vec{e})$ 产生指令 $meth:m:e:\vec{e},e$ 对应于方法调用的目标对象，返回类型非 void 的方法调用将其返回值赋予对应的变量。对象成员变量赋值 $e_1.\ f:=e_2$ 与对象成员变量访问类似，不同的是 e_1 所指对象的成员变量 f 的值被赋予 e_2 的值。条件语句和循环语句分别转换为指令 if:$S_1:S_2$ 和 while:S，规则 if 和 while 分别与之反应。语句转换函数 $\mathscr{S}[\![\_]\!]$ 的具体定义如下：

$\mathscr{S}[\![skip]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle skip\rangle,\omega\rangle\quad\mathscr{S}[\![Tx]\!]\langle a:i,\omega\rangle=\langle a:i+1,i:\langle\text{declare}(T,x)\rangle,\omega\rangle$

$\mathscr{S}[\![x:=e]\!]=Assign(x,e)\circ\xi[\![e]\!]\quad\mathscr{S}[\![x:=\text{new } C(\vec{e})]\!]=Assign(x,new:C)\circ NewObject(C,\vec{e})\circ\xi[\![\vec{e}]\!]$

$\mathscr{S}[\![x:=e.\ m(\vec{e})]\!]=Assign(x,meth:m)\circ Method(m,e,\vec{e})\circ\xi[\![e,\vec{e}]\!]$

$\mathscr{S}[\![e.\ m(\vec{e})]\!]=Method(m,e,\vec{e})\circ\xi[\![e,\vec{e}]\!]$

$\mathscr{S}[\![e_1.\ f:=e_2]\!]=FieldAssign(e_1.\ f,e_2)\circ\xi[\![e_1,e_2]\!]\quad\mathscr{S}[\![\text{return } e]\!]=\text{Return}(e)\circ\xi[\![e]\!]$

$\mathscr{S}[\![\text{if } e \text{ then } S_1 \text{ else } S_2]\!]=Cond(e,S_1,S_2)\circ\xi[\![e]\!]\quad\mathscr{S}[\![\text{while } e \text{ do } S]\!]=\text{While}(e,S)\circ\xi[\![e]\!]$

$\mathscr{S}[\![S_1;S_2]\!]=\mathscr{S}[\![S_2]\!]\circ\mathscr{S}[\![S_1]\!]$

$skip\equiv\gamma x.\ x\quad Assign(x,y)\langle a:i,\omega\rangle=\langle a:i+1,i:\langle assign(x,y)\rangle,\omega\rangle$

$assign(x,y)=\gamma(x:v,y:v').\ (x:v',y:v')\quad NewObject(C,\vec{e})\langle a:i,\omega\rangle=\langle a:i+1,i:new:C:\vec{e},\omega\rangle$

$Method(m,e,\vec{e})\langle a:i,\omega\rangle=\langle a:i+1,i:meth:m:e:\vec{e},\omega\rangle$

$FieldAssign(e_1.\ f,e_2)\langle a:i,\omega\rangle=\langle a:i+1,i:\langle fieldAssign(e_1.\ f,e_2)\rangle,\omega\rangle$

$fieldAssign(e_1.\ f,e_2)\equiv e_1:v\lhd v:C:\langle f:v',\omega\rangle,\gamma(e_1:v,e_2:v'',v:C:\langle f:v',\omega\rangle).\ v:C:\langle f:v'',\omega\rangle\rhd$

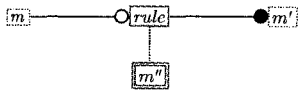
$declare(T,x)=\gamma y.\ y,x:\text{init}(T)\quad\text{Return}(e)\langle a:i,\omega\rangle=\langle a:i+1,i:\text{return}:e,\omega\rangle$

$Cond(e,S_1,S_2)\langle a:i,\omega\rangle=\langle a:n+1,i:\text{if}:S_1:S_2,\omega\rangle\quad\text{While}(e,S)\langle a:i,\omega\rangle=\langle a:i+1,i:\text{while}:e:S,\omega\rangle$

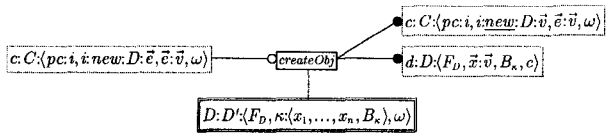
4.4 程序转换

程序转换函数映射 $\mathscr{JJ}$ 程序到分子上的函数。程序转换包含 3 个步骤：首先由 $\mathscr{C}[\![\_]\!]$ 进行类定义转换，然后由 $\mathscr{S}[\![\_]\!]$ 进行语句转换，最后由 $\mathscr{J}_{\mathscr{J}}$ 生成控制程序执行的反应规则。下

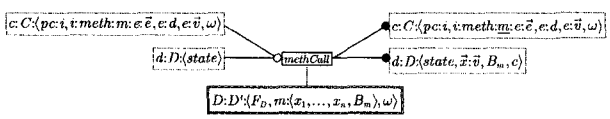
面只分析与对象有关的反应规则，其他控制相关的规则与文献[16]中的类似，不再分析。为了便于理解，反应规则可以图形化地表示如下，其中 m——○表示参与反应的分子，——● m' 表示反应生成的分子，$rule$ 为反应规则，m''……为参与反应但未发生改变的分子。



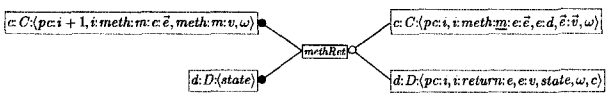
对象创建语句 $x:=\text{new } D(\vec{e})$ 转换后的指令 $i:\text{new}:D:\vec{e}$ 由规则 *createObj* 处理。*createObj* 反应产生了一个新的分子 $d:D:\langle F_D,\vec{x}:\vec{v},B_\kappa,c\rangle$ 来代表类 D 的一个新的对象实例，B_κ 为类 D 的构造函数体，$\vec{v}$ 为传递的参数值，c 代表对象创建语句所在的对象。分子 $d:D:\langle F_D,\vec{x}:\vec{v},B_\kappa,c\rangle$ 可以参与反应，当其到达稳定状态后将对象标识 d 传递给变量 x。



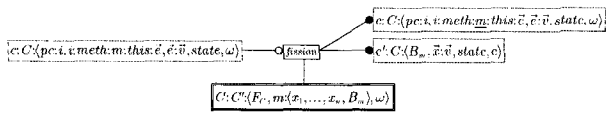
对象间方法调用 $x:=e.m(\vec{e})$ 与对象创建类似，不同的是目标对象必须已经存在，规则 *methCall* 处理方法调用指令 $i:meth:m:e:\vec{e}$。



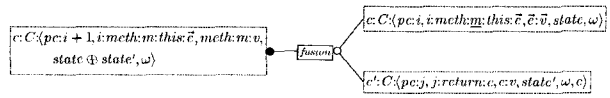
规则 *methRet* 处理方法调用返回，$d:D:\langle pc:i,i:return:e,e:v,state,\omega,c\rangle$ 表示对象 d 已经处于稳定状态，*methRet* 反应后，表达式 e 的值返回给其调用对象 c，对象 d 中只有状态部分被保留。



对象内方法调用需要特别的处理。规则 *fission* 反应产生的新分子 $c':C:\langle B_m,\vec{x}:\vec{v},state,c\rangle$，可以看作是原来分子的一个拷贝，它们具有相同的状态部分，但是其反应部分替换为被调用的方法。



当对象自身方法调用执行完成后，规则 *fusion* 将该对象与其副本合并，得到更新的对象。



程序转换的完整定义如下：

$\mathscr{PROG}=\mathscr{S}_{\mathscr{G}}(P,\mathscr{R})\circ\mathscr{S}[\![S_P]\!]\circ\mathscr{C}[\![Cdec]\!],P\equiv Cdec\ S_P$

$\mathscr{S}_{\mathscr{G}}(x,\mathscr{R})\langle\omega\rangle=\langle x:\langle\omega\rangle,\mathscr{R}\rangle\quad\mathscr{R}\equiv\mathscr{I},\mathscr{O}$

$\mathscr{B}[\![B,S,p]\!]\langle\omega\rangle=Block(B,p)\circ\mathscr{S}[\![S]\!]\quad Block(B,p)\langle\omega\rangle=B:\langle\omega,p\rangle$

$\mathscr{I}\equiv$run,if,while

run=! $\gamma(p:\langle pc:a,a:\langle r\rangle,\omega\rangle).(p:\langle pc:a+1,a:\langle r\rangle,r,\omega\rangle)$

if=! $\gamma(p:\langle pc:a,a:\text{if}:S_1:S_2,b:t,local,\omega\rangle)[t].$

$p:\langle pc:a,a:\underline{\text{if}}:S_1:S_2,\mathscr{B}[\![B,S_1,p]\!]InitSolution(local)\triangleright,local,\omega\rangle$

$[p:\langle pc:a,a:\underline{\text{if}}:S_1:S_2,\mathscr{B}[\![B,S_2,p]\!]InitSolution(local)\triangleright,local,\omega\rangle],$

$!\gamma(p:\langle pc:a,a:\underline{\text{if}}:S_1:S_2,if:\langle x\rangle,state,\omega\rangle,B:\langle pc:a,a:\text{return},local',p,\omega'\rangle).$

$p:\langle pc:a+1,a:\text{if}:S_1:S_2,local\oplus local',\omega\rangle$

while=! $\gamma(p:\langle pc:a,a:\text{while}:S,b:t,local,\omega\rangle)[t].$

$p:\langle pc:a,a:\underline{\text{while}}:S,\mathscr{B}[\![B,S,p]\!]InitSolution(local)\triangleright,local,\omega\rangle$

$[p:\langle pc:a+1,a:\text{while}:S,local,\omega\rangle],$

$!\gamma(p:\langle pc:a,a:\underline{\text{while}}:S,local,\omega\rangle,B:\langle pc:a,a:\text{return},local',p,\omega'\rangle)[t].$

$p:\langle pc:a-1,a:\text{while}:S,local\oplus local',\omega\rangle$

$\mathscr{O}\equiv createObj,methCall,methRet,fission,fusion$

$createObj=!\ \gamma(c:\omega:\langle pc:i,i:\text{new}:D:\vec{e},\vec{e}:\vec{v},\omega\rangle,D:D':\langle F_D,\kappa_D:\langle\vec{x},B\rangle,\omega\rangle).$
$c:\omega:\langle pc:i,i:\underline{\text{new}}:D:\vec{e},\vec{e}:\vec{v},\omega\rangle,d:D:\langle F_D,\vec{x}:\vec{v},B,c\rangle,D:D':\langle F_D,\kappa_D:\langle\vec{x},B\rangle,\omega\rangle$

$methCall=!\ \gamma(c:\omega:\langle pc:i,i:meth:m:e:\vec{e},e:d,\vec{e}:\vec{v},\omega\rangle,d:D:\langle F_D\rangle,D:D':\langle m:\langle\vec{x},B\rangle,\omega\rangle).$
$c:\omega:\langle pc:i,i:meth:\underline{m}:e:\vec{e},e:d,\vec{e}:\vec{v},\omega\rangle,d:D:\langle F_D,\vec{x}:\vec{v},B,c\rangle,D:D':\langle m:\langle\vec{x},B\rangle,\omega\rangle$

$methRet=!\ \gamma(c:\omega:\langle pc:i,i:meth:\underline{m}:e:\vec{e},e:d,\vec{e}:\vec{v},\omega\rangle,d:D:\langle pc:i,i:\text{return}:e,e:v,c,F_d\rangle).$
$c:\omega:\langle pc:i+1,i:meth:m:e:\vec{e},meth:m:v,\omega\rangle,d:D:\langle F_d\rangle$

$fission=!\ \gamma(c:C:\langle pc:i,i:meth:m:\text{this}:\vec{e},\vec{e}:\vec{v},F_c,\omega\rangle,C:C':\langle F_C,m:\langle\vec{x},B\rangle,\omega\rangle).$
$c:\omega:\langle pc:i,i:meth:\underline{m}:\text{this}:\vec{e},\vec{e}:\vec{v},\omega\rangle,c':C:\langle F_c,\vec{x}:\vec{v},B,c\rangle,C:C':\langle F_C,m:\langle\vec{x},B\rangle,\omega\rangle$

$fusion=!\ \gamma(c:C:\langle pc:i,i:meth:\underline{m}:\text{this}:\vec{e},\vec{e}:\vec{v},F_c,\omega\rangle,c':C:\langle F_{c'},pc:j,j:\text{return}:e,e:v,c,\omega\rangle).$
$c:\omega:\langle pc:i+1,i:meth:m:\text{this}:\vec{e},meth:m:v,F_c\oplus F_{c'},\omega\rangle$

下面的等价关系刻画了从 $\mathscr{GJ}$ 到 $\gamma_{\mathscr{RM}}$ 演算转换的正确性。

$$\mathscr{GJ}[\![P]\!](\emptyset,\emptyset)=(s,\sigma)\Leftrightarrow(\mathscr{PROG}[\![P]\!]InitSolution(\emptyset)\xrightarrow[\gamma]{*}\lfloor\langle Class_{env},P:\langle m\rangle,O,\mathscr{I},\mathscr{O}\rangle\rfloor\wedge stack(P:\langle m\rangle)=s\wedge heap(O)=\sigma)$$

其中

$\mathscr{GJ}[\![P]\!](\emptyset,\emptyset)=(s,\sigma)\Leftrightarrow CT_P\vdash\langle S_P,\emptyset,\emptyset\rangle\rightsquigarrow\langle s,\sigma\rangle$

$InitSolution(s)=\langle pc:1,a:1,s\rangle$

$stack(P:\langle local,\omega\rangle)=s$

$x:v\in local\Leftrightarrow s(x)=v$

$c:C:\langle state\rangle\in O\Leftrightarrow\sigma(c)=(C,state)$

结束语 传统化学计算模型，如 Gamma，CHAM 等，是一类基于自然现象比喻的计算模型，其他类似的计算模型包括膜计算、分子计算等。化学编程语言可以有效消除与计算逻辑本身无关的约束和人为顺序性，从而适用于系统的高层抽象描述。文献[18]使用 CHAM 形式化描述指挥控制系统的软件体系结构；文献[19]给出了生物序列拼装欧拉路径算法的 Gamma 描述和并行化实现方案；文献[20]描述了一种基于扩展 CHAM 模型的集成测试方法，软件系统及其构件

的 CHAM 描述被用于生成子系统测试用例。文献[8,9,17]综述了 Gamma 和化学计算模型在不同领域中的应用和研究进展。

最初的化学计算模型 Gamma[1,2]是一个并行程序设计模型，一个 Gamma 程序包含一个多重集和一组反应规则，程序执行是根据反应规则重写多重集。在 Gamma 程序中，多重集中的元素不能包含 Gamma 程序和反应规则，因此 Gamma 是一阶化学计算模型。高阶 Gamma[3]是对 Gamma 的高阶扩展，程序可以作为多重集中的元素，但是反应规则和多重集仍然是分离的。HOCL[5,6]是对传统化学计算模型的扩展和泛化，其最大的不同在于多重集中的元素可以不仅仅是原子值，多重集和程序都可以是其元素。

化学抽象机(CHAM)[4]同样基于化学比喻可以看作 $\gamma_{\mathcal{RA}}$ 演算的一阶部分。CHAM 被用来描述过程演算 CCS 和 π 演算的操作语义[4]。由于 π 演算可以描述面向对象语言[10]，因此给出一种描述对象系统的化学抽象机也是可能的。

Gamma 程序演算[13]，通过定义程序合成运算符可以实现特定的控制机制。例如，程序 P_1 和 P_2 的顺序合成 $P_2 \circ P_1$ 表示当程序 P_1 执行得到稳定的多重集后，程序 P_2 才开始执行。一种可能的方式是将面向对象语言转换为 Gamma 程序演算并定义恰当的合成运算符，但是必须对 Gamma 语义进行扩展。本文则是在保持已有的化学语义不变的前提下，给出从面向对象语言到 $\gamma_{\mathcal{RA}}$ 演算的转化。仅使用一阶化学计算模型，似乎很难定义合适的合成算符来描述对象间的消息传递。如何结合高阶化学计算模型和 Gamma 程序演算是我们正在研究的问题之一。

本文中的数据不仅包含简单的数值类型，也包含了可能具有复杂结构的对象类型。Gamma 的结构化扩充[14,15]通过结构多重集来刻画具有数据值关联的复杂数据对象和程序类型，如何结合结构化多重集和本文的语义框架来定义正确的静态类型系统和动态类型系统是一个有待研究的问题。

本文考虑的面向对象语言 $\mathcal{OL}$ 非常简单，仅包含了面向对象的基本特征，如何描述其他语言特征和机制，如接口、异常处理、垃圾回收等，也是一个有待研究的问题。尽管 $\mathcal{OL}$ 是顺序语言，但由于高阶化学 $\gamma_{\mathcal{RA}}$ 演算内在的并行性，因此本文给出的语义描述可以容易地扩展到并行面向对象语言。

传统化学计算模型的高层抽象性和全局性，使其缺乏模块化，且难于表示传统的控制机制，从而难于应用于大规模的环境。高阶化学计算模型的表达能力，可以有效克服这些困难。基于这一思想，本文形式地给出了简单顺序面向对象语言到高阶化学 $\gamma_{\mathcal{RA}}$ 演算的转换，即将对象看作化学分子；对象系统看作化学溶液；对象交互刻画为反应规则，从而给出了面向对象语言的一种化学语义描述，提供了一种描述面向对象系统的新视角，也为结合面向对象编程和化学编程提供了一种可能。

高阶化学模型提供了一种通用的模块化和可扩展的语义描述框架，文献[16]使用高阶化学模型描述简单命令式语言的语义，通过定义描述对象及其交互的分子和反应规则，可以直接得到简单命令式面向对象语言的语义描述。同样，当考虑新的控制机制时，可以在已有语义框架的基础上定义对应的反应规则。面向方面的编程[21](Aspect Oriented Programming)提供了一种新的结构 aspects 用于横切关注的模块化实现。面向方面编程语言通常是对已有编程语言的扩展，例如，AspectJ[22]是对 Java 的面向方面的扩展。方面和对象之间的交互是复杂而微妙的，如何在化学语义框架中描述面向方面语言的语义是下一步要研究的问题。

参 考 文 献

[1] Banatre J P, Le Metayer D. A new computational model and its discipline of programming[R]. RR0566. INRIA, 1986

[2] Banatre J P, Le Metayer D. Programming by multiset transformation[J]. Communications of the ACM(CACM), 1993, 36(1): 98-111

[3] Le Metayer D. Higher-order multiset programming [A]// Bleloch G, Chandy K, Jagannathan S, eds. Proceedings of the DIMACS workshop on specifications of parallel algorithms[C]. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 18, 1994: 179-200

[4] Berry G, Boudol G. The chemical abstract machine[J]. Theoretical Computer Science, 1992, 96: 217-248

[5] Banatre J P, Fradet P, Radenac Y. Principles of chemical programming[A]// Abdennadher S, Ringeissen C, eds. Proceedings of the 5th International Workshop on Rule-Based Programming [C]. ENTCS, 2005, 124(1): 133-147

[6] Banatre J P, Fradet P, Radenac Y. G eneralised multisets for chemical programming[J]. Mathematical Structures in Computer Science, 2006, 16(4): 557-580

[7] Banatre J P, Fradet P, Radenac Y. Classical coordination mechanisms in the chemical model [A]// Bertot Y, Huet G, Levy J J, et al. , eds. From semantics to computer science: essays in honor of Gilles Kahn [C]. Cambridge: Cambridge University Press, 2008: 29-50

[8] Banatre J P, Fradet P, Le Metayer D. Gamma and the chemical reaction model: fifteen years after [A]// Calude CS, Pun G, Rozenberg G, et al. , eds. Multiset Processing[C]. LNCS 2235, Heidelberg: Springer-Verlag, 2001: 17-44

[9] Banatre J P, Fradet P, Radenac Y. The chemical reaction model recent developments and prospects [A]// Wirsing M, Banatre JP, Holzl M, et al. , eds. Software-Intensive Systems and New Computing Paradigms: Challenges and Visions[C]. LNCS 5380, Heidelberg: Springer-Verlag, 2008: 209-234

[10] Sangiorgi D. An interpretation of typed objects into typed pi-calculus[J]. Information and Computation, 1998, 143(1): 34-73

[11] Arnold K, Gosling J, Holmes D. The Java™ Programming Language(Fourth Edition)[M]. Addison-Wesley, 2005

[12] Igarashi A, Pierce B C, Wadler P. Featherweight Java: A minimal core calculus for Java and GJ[J]. ACM Transactions on Programming Languages and Systems, 2001, 23(3): 396-450

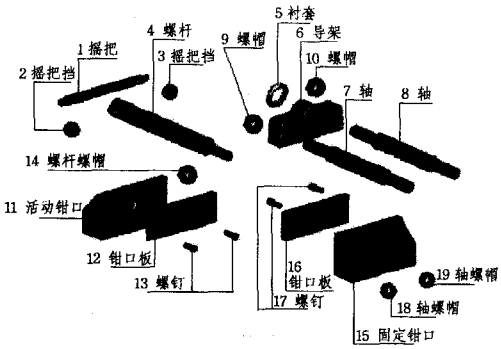
[13] Hankin C, Le Metayer D, Sands D. A calculus of gamma programs[A]// Banerjee U, Gelernter D, Nicolau A, Padua D, eds. Proc. of the 5th International Workshop on Languages and Compilers for Parallel Computing [C]. LNCS 757, Heidelberg: Springer-Verlag, 1993: 342-355

[14] Fradet P, Le Metayer D. Structured gamma [J]. Science of Computer Programming, 1998, 31(2/3): 263-289

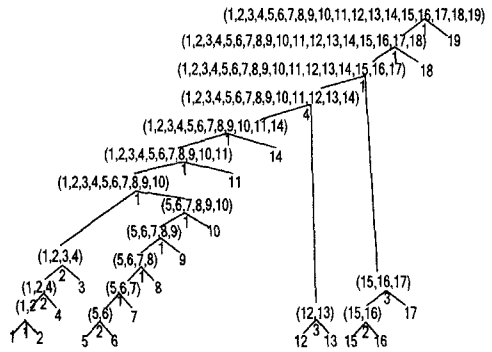
[15] 韦梓楚. 关于并行语言 Gamma 的结构化扩充[J]. 软件学报, 2000, 11(11): 1560-1566

(下转第 174 页)

平台 Windows 2000，P4 3.00GHz CPU，256MB RAM 上，以一个具有 19 个零件的机用虎钳为对象进行实验，如图 5(a)所示。得到虎钳的最优装配序列如图 5(b)所示，最优装配代价为 16，运行时间是 116.66s。如果算法基于 OBDD 实现，算法运行时间是 234.51s。基于 ZBDD 的装配序列优化算法比基于 OBDD 的装配序列优化算法节省一半的时间。



(a) 虎钳的零件图



(b) 虎钳的最优装配序列

图 5 虎钳装配实例

结束语 采用赋时 Petri 网为装配序列建模，将装配类型、装配时间以及装配难度等影响装配操作的因素统一为装配代价，体现在迁移的赋时上；然后利用赋时 Petri 网到普通 Petri 网的转换算法，将装配序列的赋时 Petri 网模型转换为等价的普通 Petri 网模型；最后给出了装配序列规划的符号 ZBDD 求解算法。算法中对状态空间的隐式搜索避免了状态和搜索的显式枚举，实现了隐式高效操作，提高了算法的执行效率，通过实例检验了算法的有效性。

对于实际应用中的大型复杂装配体，可以采用分层和子装配体局部规划的思想，将复杂装配体分解为简单子装配体，再将子装配体看作零件进行规划，这也是今后将要开展的重要研究工作。

参 考 文 献

[1] Molloy E, Yang H, Browne J. Feature-based Modeling in Design for Assembly [J]. International Journal of Computer Integrated Manufacturing, 1993, 6(12): 119-125

[2] 王俊峰，李世其，刘继红，等. 计算机辅助装配规划研究综述[J]. 工程图学学报，2005，26(2)：1-6

[3] de Meool L S H. Representation of Mechanical Assembly Sequences [J]. IEEE Transaction on Robotics and Automation, 1991, 7(2): 211-227

[4] Gottipolu B, GHhosh K. A Simplified and Efficient Representation for Evaluation and Selection of Assembly Sequences [J]. Computers in Industry, 2003, 50(2): 251-264

[5] Fazio D T, Whitney D E. Simplified Generation of All Mechanical Assembly Sequences [J]. IEEE Journal Robotics and Automation, 1987, 3(6): 640-658

[6] de Mello L S H, Sanderson A C. A Correct and Complete Algorithm for Mechanical Assembly Sequences [J]. IEEE Transaction on Robotics and Automation, 1991, 7(2): 228-240

[7] Tianlong G, Zhoubo X, Zhifei Y. Symbolic OBDD Representations for Mechanical Assembly Sequences [J]. Computer-Aided Design, 2008, 40(4): 411-421

[8] Lazzerini B, Marcelloni F. A Genetic Algorithm for Generating Optimal Assembly plans [J]. Artificial Intelligence in Engineering, 2000, 14(4): 319-329

[9] 彭涛，李世真，王俊峰，等. 基于集成干涉矩阵的蚁群装配序列规划 [J]. 计算机科学，2010，37(4)：179-182

[10] 古天龙，徐周波. 有序二叉决策图及应用[M]. 北京：科学出版社，2009

[11] Minato S. Zero-suppressed BDDs and Their Applications [J]. International Journal on Software Tools for Technology Transfer, 2001, 3(2): 156-170

[12] 李凤英，古天龙，徐周波. Petri 网的符号 ZBDD 可达树分析技术 [J]. 计算机学报，2009，32(12)：2420-2428

[13] Luiz S, Homen D M, Sanderson A C. AND/OR Graph Representation of Assembly Plans [J]. IEEE Transactions on Robotics and Automation, 1990, 6(2): 188-198

[14] Xiaoming Z, Pingan D, Yuege Z. A Model-based Approach to Assembly Sequence Planning [C] // Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation. 2007: 599-606

[15] Somenzi F. CUDD: CU Decision Diagram Package Release 2.3.1 [EB/OL]. http://vlsi.Colorado.edu/~fabio/CUDD/cuddIntro.html, 2001-02-16

(上接第 142 页)

[16] 闫伟，卢炎生. 命令式语言的化学语义[J]. 小型微型计算机系统

[17] 黄林鹏，童维勤. 并行计算 GAMMA 概述[J]. 计算机科学，1994，21(5)：20-24

[18] 赵恒，王振宇，曹万华，等. 化学抽象机的分析与应用研究[J]. 计算机科学，2003，30(1)：42-45

[19] 廖文昭，童维勤，蔡立志. 生物序列拼装欧拉路径算法的 Gamma 描述及其并行化研究[J]. 小型微型计算机系统，2004，25(4)：707-711

[20] 叶俊民，罗景，朱凯，等. 基于扩展 CHAM 模型的软件集成测试方法[J]. 计算机科学，2005，32(6)：199-201

[21] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-Oriented Programming[A] // Aksit M, Matsuoka S, eds. Proceedings of the 11th European Conference on Object-Oriented Programming [C]. LNCS 1241, Heidelberg: Springer-Verlag, 1997: 220-242

[22] Kiczales G, Hilsdale E, Hugunin J, et al. An Overview of Aspect][A]//Knudsen J L, ed. Proceedings of the 15th European Conference on Object-Oriented Programming[C]. LNCS 2072, Heidelberg: Springer-Verlag, 2001: 327-353