Algebraic structure for untyped Racket

Eric Griffis
dedbox@gmail.com

February 3, 2019

Abstract

A language for defining simple macros and pure functions with pattern-based syntax, along with first-class data constructors. Type system not included.

1 The core model

$$
\begin{array}{rclr}
t & ::= & t\ t & \text{application} \\
 & | & t; t & \text{sequence} \\
 & | & \mu p.t & \text{macro clause} \\
 & | & \phi p.t & \text{function clause} \\
 & | & x & \text{variable} \\
 & | & \delta & \text{constructor} \\
 & | & \diamond & \text{null} \\
v & ::= & \mu p.t; \cdot & \text{macro} \\
 & | & \phi p.t; \cdot & \text{function} \\
 & | & \delta & \text{constructor} \\
 & | & \delta\ (v; \cdot) & \text{instance} \\
 & | & \diamond & \text{null}
\end{array}
\qquad
\begin{array}{rclr}
p & ::= & p\ p & \text{application} \\
 & | & p; p & \text{sequence} \\
 & | & \_ & \text{wildcard} \\
 & | & x & \text{variable} \\
 & | & \delta & \text{constructor} \\
 & | & \diamond & \text{null}
\end{array}
$$

A *constructor* (δ) identifies a data type. Constructors are first class; they can be matched against, collected into lists, and passed between functions. A constructor applied to a value is an *instance* of the denoted type.

$$
\frac{t_1 \rightsquigarrow t_1'}{t_1; t_2 \rightsquigarrow t_1'; t_2}\ \textsc{Seq1}
\qquad\qquad
\frac{t_2 \rightsquigarrow t_2'}{v_1; t_2 \rightsquigarrow v_1; t_2'}\ \textsc{Seq2}
$$

Sequences ($t; t$) are evaluated eagerly from left to right. Sequences collect multiple clauses into a single macro or function. A sequence is *uniform* ($t; \cdot$) when every sub-term has the same shape.

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\ t_2 \rightsquigarrow t_1'\ t_2}\ \textsc{App1} \qquad \frac{p_{11} \times t_2 = \sigma \qquad \sigma(t_{12}) = t_{12}'}{(\mu p_{11}.t_{12})\ t_2 \rightsquigarrow t_{12}'}\ \textsc{AppM}$$

$$\frac{v_1 \not\sim \mu p.t;\cdot \qquad t_2 \rightsquigarrow t_2'}{v_1\ t_2 \rightsquigarrow v_1\ t_2'}\ \textsc{App2} \qquad \frac{p_{11} \times v_2 = \sigma \qquad \sigma(t_{12}) = t_{12}'}{(\phi p_{11}.t_{12})\ v_2 \rightsquigarrow t_{12}'}\ \textsc{AppF}$$

A *macro clause* ($\mu p.t$) is a lambda abstraction that can reject or deconstruct a term by pattern matching against its unevaluated argument. A *macro* ($\mu p.t;\cdot$) is a macro clause or a uniform sequence of macro clauses. A macro gets stuck if every clause rejects the argument term.

A *function clause* ($\phi p.t$) is a lambda abstraction that can reject or deconstruct a value by pattern matching against its fully-evaluated argument. A *function* ($\phi p.t;\cdot$) is a function clause or a uniform sequence of function clauses. A function gets stuck if every clause rejects the argument value.

An *application* ($t\ t$) is evaluated quasi-eagerly, starting on the left. If the left side reduces to a macro, it is a *macro application* and its patterns are matched against the un-evaluated right side. Otherwise, the right side is evaluated; if the left side reduced to a function, it is a *function application* and its patterns are matched against the evaluation result. In either case, if the match succeeds, any bound pattern variables are substituted in the consequent term, which becomes the result of the application.

$$\frac{(\mu p_1.t_2)\ t_4 \rightsquigarrow t_2'}{(\mu p_1.t_2; v_3)\ t_4 \rightsquigarrow t_2'}\ \textsc{Mac1} \qquad \frac{p_1 \times t_4 = \diamond}{(\mu p_1.t_2; v_3)\ t_4 \rightsquigarrow v_3\ t_4}\ \textsc{Mac2}$$

$$\frac{(\phi p_1.t_2)\ v_4 \rightsquigarrow t_2'}{(\phi p_1.t_2; v_3)\ v_4 \rightsquigarrow t_2'}\ \textsc{Fun1} \qquad \frac{p_1 \times v_4 = \diamond}{(\phi p_1.t_2; v_3)\ v_4 \rightsquigarrow v_3\ v_4}\ \textsc{Fun2}$$

The clauses of a uniform sequence are applied in order, from left to right. The first successful match determines the result.

1.1 Pattern matching

$$\frac{p_1 \times t_1 = \sigma_1 \qquad p_2 \times t_2 = \sigma_2}{(p_1\ p_2) \times (t_1\ t_2) = \sigma_1 \cup \sigma_2} \qquad \frac{p_1 \times t_1 = \sigma_1 \qquad p_2 \times t_2 = \sigma_2}{(p_1; p_2) \times (t_1; t_2) = \sigma_1 \cup \sigma_2} \qquad x_1 \times t_2 = \{x_1 \mapsto t_2\}$$

$$\frac{\delta_1 = \delta_2}{\delta_1 \times \delta_2 = \{\}} \qquad \_ \times t_1 = \{\} \qquad \diamond \times \diamond = \{\}$$

A *constructor pattern* (δ) matches any constructor with the same name. A *null pattern* ($\diamond$) matches only the null term. A *wildcard* ($\_$) matches anything. A *variable* (x) matches anything and binds itself to the matched term. An *application pattern* $(p\ p)$ or *sequence pattern* $(p;p)$ matches its sub-patterns against the sub-terms of an application and passes along any variable bindings.

1.2 Variable substitution

$$\sigma(t_1\ t_2) = \sigma(t_1)\ \sigma(t_2) \qquad \sigma(t_1;t_2) = \sigma(t_1);\sigma(t_2) \qquad \frac{\sigma' = \sigma \setminus \mathrm{vars}(p_1) \qquad t_2' = \sigma'(t_2)}{\sigma(\mu p_1.t_2) = \mu p_1.t_2'}$$

$$\frac{\sigma' = \sigma \setminus \mathrm{vars}(p_1) \qquad t_2' = \sigma'(t_2)}{\sigma(\phi p_1.t_2) = \phi p_1.t_2'} \qquad \frac{(x_1 \mapsto t) \in \sigma}{\sigma(x_1) = t} \qquad \sigma(\delta_1) = \delta_1 \qquad \sigma(\diamond) = \diamond$$

1.3 Examples

1.3.1 Numbers

$$\mathrm{add} = \phi(a\ \mathrm{Zero}).a; \phi(a\ (\mathrm{Succ}\ b)).\mathrm{Succ}\ (\mathrm{add}\ (a\ b))$$

add ((Succ Zero) (Succ (Succ Zero)))
$\rightsquigarrow$ Succ (add ((Succ Zero) (Succ Zero)))
$\rightsquigarrow$ Succ (Succ (add ((Succ Zero) Zero)))
$\rightsquigarrow$ Succ (Succ (Succ Zero))

$$\mathrm{mul} = \phi(a\ \mathrm{Zero}).\mathrm{Zero}; \phi(a\ (\mathrm{Succ}\ b)).\mathrm{add}\ (a\ (\mathrm{mul}\ (a\ b)))$$

Denote by $N \in \mathbb{N}$ a series of N Succ s and a Zero.

mul (2 3)
$\rightsquigarrow$ add (2 (mul (2 2)))
$\rightsquigarrow$ add (2 (add (2 (mul (2 1)))))
$\rightsquigarrow$ add (2 (add (2 (add (2 (mul (2 0)))))))
$\rightsquigarrow$ add (2 (add (2 (add (2 0)))))
$\rightsquigarrow$ add (2 (add (2 2)))
$\rightsquigarrow$ add (2 (Succ (add (2 1))))
$\rightsquigarrow$ add (2 (Succ (Succ (add (2 0)))))
$\rightsquigarrow$ add (2 4)
$\rightsquigarrow$ Succ (add (2 3))
$\rightsquigarrow$ Succ (Succ (add (2 2)))
$\rightsquigarrow$ Succ (Succ (Succ (add (2 1))))
$\rightsquigarrow$ Succ (Succ (Succ (Succ (add (2 0)))))
$\rightsquigarrow$ 6

1.3.2 Booleans

$$\begin{aligned}
\mathrm{not} &= \phi\mathrm{False}.\mathrm{True}; \phi_.\mathrm{False} \\
\mathrm{and} &= \mu(a\ b).(\phi\mathrm{False}.\mathrm{False}; \phi_.b)\ a \\
\mathrm{or} &= \mu(a\ b).(\phi\mathrm{False}.b; \phi x.x)\ a \\
\mathrm{xor} &= \mu(a\ b).(\phi\mathrm{False}.b; \phi x.\mathrm{and}\ ((\mathrm{not}\ b)\ x))\ a
\end{aligned}$$

or ((not True) (and ((xor (True True)) True)))
$\rightsquigarrow$ or (False (and ((xor (True True)) True)))
$\rightsquigarrow$ and ((xor (True True)) True)
$\rightsquigarrow$ and ((and ((not True) True)) True)
$\rightsquigarrow$ and ((and (False True)) True)
$\rightsquigarrow$ and (False True)
$\rightsquigarrow$ False

1.3.3 Lists

$$\mathrm{list} = \mu(x\ \diamond).\mathrm{Cons}\ (x\ \mathrm{Nil}); \mu(x\ xs).\mathrm{Cons}\ (x\ (\mathrm{list}\ xs))$$

list (1 (2 (3 $\diamond$)))
$\rightsquigarrow$ Cons (1 (list (2 (3 $\diamond$))))
$\rightsquigarrow$ Cons (1 (Cons (2 (list (3 $\diamond$)))))
$\rightsquigarrow$ Cons (1 (Cons (2 (Cons (3 Nil)))))

$$\begin{aligned}
\mathrm{rev} &= \phi(\mathrm{Nil}\ a).a; \phi((\mathrm{Cons}\ (y\ ys))\ a).\mathrm{rev}\ (ys\ (\mathrm{Cons}\ (y\ a))) \\
\mathrm{reverse} &= \phi xs.\mathrm{rev}\ (xs\ \mathrm{Nil})
\end{aligned}$$

reverse (Cons (1 (Cons (2 (Cons (3 Nil))))))
$\rightsquigarrow$ rev ((Cons (1 (Cons (2 (Cons (3 Nil)))))) Nil)
$\rightsquigarrow$ rev ((Cons (2 (Cons (3 Nil)))) (Cons (1 Nil)))
$\rightsquigarrow$ rev ((Cons (3 Nil)) (Cons (2 (Cons (1 Nil)))))
$\rightsquigarrow$ rev (Nil (Cons (3 (Cons (2 (Cons (1 Nil)))))))
$\rightsquigarrow$ Cons (3 (Cons (2 (Cons (1 Nil)))))

$$\mathrm{append} = \phi(\mathrm{Nil}\ ys).ys; \phi((\mathrm{Cons}\ (x\ xs))\ ys).\mathrm{Cons}\ (x\ (\mathrm{append}\ (xs\ ys)))$$

append ((Cons (1 (Cons (2 Nil)))) (Cons (3 (Cons (4 Nil)))))
$\rightsquigarrow$ Cons (1 (append ((Cons (2 Nil)) (Cons (3 (Cons (4 Nil)))))))

$\rightsquigarrow$ Cons (1 (Cons (2 (append (Nil (Cons (3 (Cons (4 Nil))))))))))
$\rightsquigarrow$ Cons (1 (Cons (2 (Cons (3 (Cons (4 Nil)))))))

$$\text{map} = \phi(\_\ \text{Nil}).\text{Nil}; \phi(f\ (\text{Cons}\ (x\ xs))).\text{Cons}\ ((f\ x)\ (\text{map}\ (f\ xs)))$$

map (Succ (Cons (3 (Cons (2 (Cons (1 Nil)))))))
$\rightsquigarrow$ Cons (4 (map (Succ (Cons (2 (Cons (1 Nil)))))))
$\rightsquigarrow$ Cons (4 (Cons (3 (map (Succ (Cons (1 Nil)))))))
$\rightsquigarrow$ Cons (4 (Cons (3 (Cons (2 (map (Succ Nil)))))))
$\rightsquigarrow$ Cons (4 (Cons (3 (Cons (2 Nil)))))

2 The extended model

2.1 Extended syntax

$$t_1\ t_2\ \cdots\ t_n \rightsquigarrow t_1\ (t_2\ \cdots\ t_n) \qquad n \geq 2 \qquad p_1\ p_2\ \cdots\ p_n \rightsquigarrow p_1\ (p_2\ \cdots\ p_n)$$

$$t_1; t_2; \cdots; t_n \rightsquigarrow t_1; (t_2; \cdots; t_n) \qquad p_1; p_2; \cdots; p_n \rightsquigarrow p_1; (p_2; \cdots; p_n)$$

$$\mu \left[\begin{array}{c|c} p_1 & t_1 \\ p_2 & t_2 \\ \vdots & \vdots \\ p_n & t_n \end{array}\right] \rightsquigarrow \mu p_1.t_1; \mu \left[\begin{array}{c|c} p_2 & t_2 \\ \vdots & \vdots \\ p_n & t_n \end{array}\right] \qquad \mu\left[\begin{array}{c|c} p_1 & t_1 \end{array}\right] \rightsquigarrow \mu p_1.t_1$$

$$\phi \left[\begin{array}{c|c} p_1 & t_1 \\ p_2 & t_2 \\ \vdots & \vdots \\ p_n & t_n \end{array}\right] \rightsquigarrow \phi p_1.t_1; \phi \left[\begin{array}{c|c} p_2 & t_2 \\ \vdots & \vdots \\ p_n & t_n \end{array}\right] \qquad \phi\left[\begin{array}{c|c} p_1 & t_1 \end{array}\right] \rightsquigarrow \phi p_1.t_1$$

2.2 Let bindings

$$\text{fix} = \phi f.(\phi x.f\ \phi y.(x\ x)\ y)\ (\phi x.f\ \phi y.(x\ x)\ y)$$

$$\text{let} \approx \mu \left[\begin{array}{cc|c} (p\ t) & \mathit{body} & (\phi p.\mathit{body})\ t \\ (p\ t; cs) & \mathit{body} & \text{let}\ (p\ t)\ \text{let}\ cs\ \mathit{body} \end{array}\right]$$

$$\text{letrec} \approx \mu \left[\begin{array}{cc|c} (p\ t) & \mathit{body} & \text{let}\ (p\ \text{fix}\ \phi p.\mathit{body})\ t \\ (p\ t; cs) & \mathit{body} & \text{letrec}\ (p\ t)\ \text{letrec}\ cs\ \mathit{body} \end{array}\right]$$

2.3 Examples

2.3.1 Numbers

$$\text{add} = \phi\left[\begin{matrix} a & \text{Zero} \\ a & \text{Succ } b \end{matrix}\,\middle|\,\begin{matrix} a \\ \text{Succ add } a\ b \end{matrix}\right] \qquad \text{mul} = \phi\left[\begin{matrix} a & \text{Zero} \\ a & \text{Succ } b \end{matrix}\,\middle|\,\begin{matrix} \text{Zero} \\ \text{add } a \text{ mul } a\ b \end{matrix}\right]$$

add (Succ Zero) Succ Succ Zero
$\rightsquigarrow$ Succ add (Succ Zero) Succ Zero
$\rightsquigarrow$ Succ Succ add (Succ Zero) Zero
$\rightsquigarrow$ Succ Succ Succ Zero

mul 2 3
$\rightsquigarrow$ add 2 mul 2 2
$\rightsquigarrow$ add 2 add 2 mul 2 1
$\rightsquigarrow$ add 2 add 2 add 2 mul 2 0
$\rightsquigarrow$ add 2 add 2 add 2 0
$\rightsquigarrow$ add 2 add 2 2
$\rightsquigarrow$ add 2 Succ add 2 1
$\rightsquigarrow$ add 2 Succ Succ add 2 0
$\rightsquigarrow$ add 2 4
$\rightsquigarrow$ Succ add 2 3
$\rightsquigarrow$ Succ Succ add 2 2
$\rightsquigarrow$ Succ Succ Succ add 2 1
$\rightsquigarrow$ Succ Succ Succ Succ add 2 0
$\rightsquigarrow$ 6

2.3.2 Booleans

$$\text{not} = \phi\left[\begin{matrix} \text{False} \\ \_ \end{matrix}\,\middle|\,\begin{matrix} \text{True} \\ \text{False} \end{matrix}\right] \qquad \text{and} = \mu(a\ b).\phi\left[\begin{matrix} \text{False} \\ \_ \end{matrix}\,\middle|\,\begin{matrix} \text{False} \\ b \end{matrix}\right]\ a$$

$$\text{or} = \mu(a\ b).\phi\left[\begin{matrix} \text{False} \\ x \end{matrix}\,\middle|\,\begin{matrix} b \\ x \end{matrix}\right]\ a \qquad \text{xor} = \mu(a\ b).\phi\left[\begin{matrix} \text{False} \\ x \end{matrix}\,\middle|\,\begin{matrix} b \\ \text{and (not } b)\ x \end{matrix}\right]\ a$$

or (not True) and (xor True True) True
$\rightsquigarrow$ or False and (xor True True) True
$\rightsquigarrow$ and (xor True True) True
$\rightsquigarrow$ and (and (not True) True) True
$\rightsquigarrow$ and (and False True) True
$\rightsquigarrow$ and False True
$\rightsquigarrow$ False

2.3.3 Lists

$$\text{list} = \mu\left[\begin{matrix} x & \diamond \\ x & xs \end{matrix}\,\middle|\,\begin{matrix} \text{Cons } (x; \text{Nil}) \\ \text{Cons } (x; \text{list } xs) \end{matrix}\right]$$

list 1 2 3 $\diamond$
$\rightsquigarrow$ Cons (1; list 2 3 $\diamond$)

$\rightsquigarrow$ Cons $(1; \text{Cons } (2; \text{list } 3)) \diamond$
$\rightsquigarrow$ Cons $(1; \text{Cons } (2; \text{Cons } (3; \text{Nil})))$

$$\text{reverse} = \phi xs.\text{letrec}\left(rev\ \phi\left[\begin{array}{cc|c} \text{Nil} & a & a \\ (\text{Cons } (y; ys)) & a & rev\ ys\ \text{Cons } (y; a) \end{array}\right]\right)\ rev\ xs\ \text{Nil}$$

reverse Cons $(1; \text{Cons } (2; \text{Cons } (3; \text{Nil})))$
$\rightsquigarrow$ rev (Cons $(1; \text{Cons } (2; \text{Cons } 3 \text{ Nil})))$ Nil
$\rightsquigarrow$ rev (Cons $(2; \text{Cons } (3; \text{Nil})))$ Cons $(1; \text{Nil})$
$\rightsquigarrow$ rev (Cons $(3; \text{Nil}))$ Cons $(2; \text{Cons } (1; \text{Nil}))$
$\rightsquigarrow$ rev Nil Cons $(3; \text{Cons } (2; \text{Cons } (1; \text{Nil})))$
$\rightsquigarrow$ Cons $(3; \text{Cons } (2; \text{Cons } (1; \text{Nil})))$

$$\text{append} = \phi\left[\begin{array}{cc|c} \text{Nil} & ys & ys \\ (\text{Cons } (x; xs)) & ys & \text{Cons } (x; \text{append } xs\ ys) \end{array}\right]$$

append (Cons $(1; \text{Cons } (2; \text{Nil})))$ Cons $(3; \text{Cons } (4; \text{Nil}))$
$\rightsquigarrow$ Cons $(1; \text{append } (\text{Cons } (2; \text{Nil})) \text{ Cons } (3; \text{Cons } (4; \text{Nil})))$
$\rightsquigarrow$ Cons $(1; \text{Cons } (2; \text{append Nil Cons } (3; \text{Cons } (4; \text{Nil}))))$
$\rightsquigarrow$ Cons $(1; \text{Cons } (2; \text{Cons } (3; \text{Cons } (4; \text{Nil}))))$

$$\text{map} = \phi\left[\begin{array}{cc|c} \_ & \text{Nil} & \text{Nil} \\ f & \text{Cons } (x; xs) & \text{Cons } (f\ x; \text{map } f\ xs) \end{array}\right]$$

map Succ Cons $(3; \text{Cons } (2; \text{Cons } (1; \text{Nil})))$
$\rightsquigarrow$ Cons $(4; \text{map Succ Cons } (2; \text{Cons } (1; \text{Nil})))$
$\rightsquigarrow$ Cons $(4; \text{Cons } (3; \text{map Succ Cons } (1; \text{Nil})))$
$\rightsquigarrow$ Cons $(4; \text{Cons } (3; \text{Cons } (2; \text{map Succ Nil})))$
$\rightsquigarrow$ Cons $(4; \text{Cons } (3; \text{Cons } (2; \text{Nil})))$

3 Host integration

3.1 Primitive data

$$t ::= \ldots \mid \ell \qquad p ::= \ldots \mid \ell \qquad v ::= \ldots \mid \ell \qquad \ell ::= \diamond$$

$$\frac{\ell_1 = \ell_2}{\ell_1 \times \ell_2 = \{\}}$$

3.1.1 Procedures

$$t ::= \ldots \mid f \qquad\qquad p ::= \ldots \mid f \qquad\qquad v ::= \ldots \mid f\ \ell \qquad\qquad f ::= \ldots$$

$$\frac{\mathrm{call}(f_1, \ell_{21}, \ell_{22}, \ldots, \ell_{2n}) = \ell'_1, \ell'_2, \ldots, \ell'_m \qquad n, m \geq 1}{f_1\ \ell_{21}\ \ell_{22}\ \cdots\ \ell_{2n} \rightsquigarrow \ell'_1\ \ell'_2\ \cdots\ \ell'_m}\ \textsc{Pro}$$

3.1.2 Numbers

$$\ell ::= \ldots \mid n \qquad\qquad\qquad f ::= \ldots \mid \texttt{=} \mid \texttt{>} \mid \texttt{<} \mid \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/}$$

$$\mathrm{fib} = \phi \left[\begin{array}{c|c} n \text{ if } \texttt{<}\ n\ 2 & 1 \\ n & \texttt{+}\ (\mathrm{fib}\ \texttt{-}\ n\ 1)\ \mathrm{fib}\ \texttt{-}\ n\ 2 \end{array} \right]$$

fib 3
$\rightsquigarrow$ + (fib - 3 1) fib - 3 2
$\rightsquigarrow$ + (fib 2) fib - 3 2
$\rightsquigarrow$ + (+ (fib - 2 1) fib - 2 2) fib - 3 2
$\rightsquigarrow$ + (+ (fib 1) fib - 2 2) fib - 3 2
$\rightsquigarrow$ + (+ 1 fib −2 2) fib - 3 2
$\rightsquigarrow$ + (+ 1 fib 0) fib - 3 2
$\rightsquigarrow$ + (+ 1 1) fib - 3 2
$\rightsquigarrow$ + 2 fib - 3 2
$\rightsquigarrow$ + 2 fib 1
$\rightsquigarrow$ + 2 1
$\rightsquigarrow$ 3

3.1.3 Booleans

$$\ell ::= \ldots \mid \top \mid \bot$$

$$\mathrm{not} = \phi \left[\begin{array}{c|c} \bot & \top \\ \text{\_} & \bot \end{array} \right] \qquad \mathrm{and} = \mu(a\ b).\phi \left[\begin{array}{c|c} \bot & \bot \\ \text{\_} & b \end{array} \right] a \qquad \mathrm{or} = \mu(a\ b).\phi \left[\begin{array}{c|c} \bot & b \\ x & x \end{array} \right] a$$

$$\mathrm{xor} = \mu(a\ b).\phi \left[\begin{array}{c|c} \bot & b \\ x & \mathrm{and}\ (\mathrm{not}\ b)\ x \end{array} \right] a$$

or (not $\top$) and (xor $\top$ $\top$) $\top$
$\rightsquigarrow$ or $\bot$ and (xor $\top$ $\top$) $\top$
$\rightsquigarrow$ and (xor $\top$ $\top$) $\top$

$\rightsquigarrow$ and (and (not $\top$) $\top$) $\top$
$\rightsquigarrow$ and (and $\bot$ $\top$) $\top$
$\rightsquigarrow$ and $\bot$ $\top$
$\rightsquigarrow$ $\bot$

3.1.4 Lists

$$\ell ::= \ldots \mid \ell\ \ell$$

3.2 Pattern Guards

$$p ::= \ldots \mid p \text{ if } t$$

$$\frac{p_1 \times t_3 = \sigma_1 \qquad \sigma_1(t_2) \rightsquigarrow^* v_2 \neq \bot}{(p_1 \text{ if } t_2) \times t_3 = \sigma_1}$$