

ATI Radeon 8500 OpenGL Fragment Shader:
An Operational Semantics Description

Aaron Lefohn Volker Daum

May 9, 2002

1 Notation Conventions

- Only OpenGL calls that define an ATI fragment shader are considered. No C/C++ syntax is included.
- Lists of unbounded size are defined recursively.
- Lists of finite size are defined using '...' between initial and final elements. Elements 0 and 1 will be explicitly declared if the list must contain at least 1 element, otherwise it is assumed that the list is allowed to be empty.

2 Pre-Defined Constant Values

n_p	=	<i>NumAllowedPasses</i>	=	2
n_o	=	<i>NumOpsPerPass</i>	=	8
n_c	=	<i>NumConstRegisters</i>	=	7
n_r	=	<i>NumRegisters</i>	=	6
n_t	=	<i>NumTextureUnits</i>	=	6

3 Reference Grammar

Pass	::=	Begin ConstList PassList End
Begin	::=	<code>glBeginFragmentShaderATI()</code> ;
End	::=	<code>glEndFragmentShaderATI()</code> ;
ConstList	::=	ConstSet ConstList
ConstSet	::=	<code>glSetFragmentShaderConstantATI(ConstDst, FloatList)</code> ;
ConstDst	::=	ConstRegister
ConstRegister	::=	<code>GL_CON_x_ATI</code> where $x = 0 \dots (n_c - 1)$
FloatList	::=	<code>GLfloat c-styleIdentifier [4]</code> = { FloatVal ₀ , FloatVal ₁ , FloatVal ₂ , FloatVal ₃ };
FloatVal	::=	literal, real number $\in [0, 1]$
PassList	::=	Pass ₀ ... Pass _(n_p-1)
Pass	::=	Sample Op
Sample	::=	M _{s₀} M _{s₁} ... M _{s_(n_r-1)}
M _s	::=	AddrExpr DataExpr

AddrExpr ::= glPassTexCoordATI(Dst, Coord, Swizzle);
 DataExpr ::= glSampleMapATI(Dst, Coord, Swizzle);
 Dst ::= Register
 Coord ::= Texture
 | Register
 Register ::= GL_REG_ x _ATI
 where $x = 0 \dots (n_r - 1)$
 Texture ::= GL_TEXTURE x _ATI
 where $x = 0 \dots (n_t - 1)$
 Swizzle ::= GL_SWIZZLE_STR_ATI
 | GL_SWIZZLE_STR_ATI
 | GL_SWIZZLE_STQ_ATI
 | GL_SWIZZLE_STR_DR_ATI
 | GL_SWIZZLE_STQ_DQ_ATI
 Op ::= $M_{o_0} \dots M_{o_{(n_o-1)}}$
 M_o ::= $M_c M_a$
 M_c ::= ColorExpr
 M_a ::= AlphaExpr
 ColorExpr ::= glColorFragmentOp x ATI(OpCode, DstSpec, SrcSpec[x];
 where $x = 1 \dots 3$
 AlphaExpr ::= glAlphaFragmentOp x ATI(OpCode, Dst, DstMod, SrcSpec[x];
 where $x = 1 \dots 3$
 OpCode ::= GL_MOV_ATI
 | GL_ADD_ATI
 | GL_MUL_ATI
 | GL_SUB_ATI
 | GL_DOT2_ADD_ATI
 | GL_DOT3_ATI
 | GL_DOT4_ATI
 | GL_MAD_ATI
 | GL_LERP_ATI
 | GL_CND_ATI
 | GL_CND0_ATI
 DstSpec ::= Dst, DstMask, DstMod
 SrcSpec ::= Src, SrcRep, SrcMod

```

DstMask      ::=  GL_NONE
                |  DstMaskBitList

DstMaskBitList ::=  DstMaskBit | ... | DstMaskBit

DstMaskBit   ::=  GL_RED_BIT_ATI
                |  GL_GREEN_BIT_ATI
                |  GL_BLUE_BIT_ATI
                |  GL_ALPHA_BIT_ATI

DstMod       ::=  GL_NONE
                |  DstModOp
                |  DstModSat
                |  DstModOp | DstModSat

DstModOp     ::=  GL_2X_BIT_ATI
                |  GL_4X_BIT_ATI
                |  GL_8X_BIT_ATI
                |  GL_HALF_BIT_ATI
                |  GL_QUARTER_BIT_ATI
                |  GL_EIGHTH_BIT_ATI

DstModSat    ::=  GL_SATURATE_BIT_ATI

SrcRep       ::=  GL_NONE
                |  GL_RED
                |  GL_GREEN
                |  GL_BLUE
                |  GL_ALPHA

SrcMod       ::=  GL_NONE
                |  SrcModOpList

SrcModOpList ::=  SrcModOp | ... | SrcModOp

SrcModOp     ::=  GL_2X_BIT_ATI
                |  GL_COMP_BIT_ATI
                |  GL_NEGATE_BIT_ATI
                |  GL_BIAS_BIT_ATI

Src          ::=  Register
                |  ConstRegister
                |  GL_ZERO
                |  GL_ONE
                |  GL_PRIMARY_COLOR_ARB
                |  GL_SECONDARY_INTERPOLATOR_ATI

```

4 Semantic Considerations

- Every register used in the operations stage of a pass must be set in the sample stage of that same pass.
- Registers may be set only once in each Pass's Sample stage.
- The last pass in the program must write to `GL_REG_0_ATI`.

- Constants declared outside of a Begin/End pair are part of the global environment, whereas constants declared between a Begin/End pair are part of the shader's local environment. Locally declared constants shadow globally defined ones.
- If the OpCode in AlphaExpr is GL_DOT4_ATI, GL_DOT3_ATI, or GL_DOT2_ADD_ATI, the expression must immediately follow a ColorExpr with the same OpCode. The parameters are ignored, the arguments to the previous ColorExpr are used, and the result is written to the alpha channel of the register specified by the Dst argument.
- If the OpCode of a ColorExpr is GL_DOT4_ATI, the only AlphaExpr that is allowed to immediately follow it is one with the same OpCode.
- Coord is only allowed to be a Texture in the Sample stage of the first Pass. In later passes, Coord may be either a Register or Texture.

5 Evaluation

Ps ::= Pass

Σ_{GL} ::= global (OpenGL) state function $\{\langle R, V \rangle, \dots\}$,
 where Σ_{GL} defines variables that are globally defined in the OpenGL machine.
 Σ_{GL} must be fully initialized before it is passed into the program. It is read-only.

R = {GL_TEXTURE0_ARB, ... GL_TEXTURE($n_t - 1$)_ARB,
 GL_REG_0_ATI, ... GL_REG_($n_r - 1$)_ATI,
 GL_CON_0_ATI, ... GL_CON_($n_c - 1$)_ATI,
 GL_SECONDARY_INTERPOLATOR_ATI,
 GL_PRIMARY_COLOR_ARB}

V = vector of length 4

Σ_s ::= local (shader) state function $\{\langle R, V \rangle, \dots\}$,
 where R = GL_REG_0_ATI, ... GL_REG_($n_r - 1$)_ATI and
 GL_CON_0_ATI ... GL_CON_($n_c - 1$)_ATI
 V = vector of length 4

5.1 Basic Program Execution

$\Sigma_{GL} \vdash \langle \text{Begin ConstSet}_0 \dots \text{ConstSet}_{(n_c-1)} \text{Ps}_0 \dots \text{Ps}_{(n_p-1)} \text{End}, \Sigma_s \rangle \mapsto_p \Sigma_{GL} \vdash \langle \text{Ps}_0 \dots \text{Ps}_{(n_p-1)}, \Sigma'_s \rangle$
 where $\langle \text{ConstSet}_0 \dots \text{ConstSet}_{(n_c-1)}, \Sigma_s \rangle \rightarrow_{cs} \langle \emptyset, \Sigma'_s \rangle$

$\Sigma_{GL} \vdash \langle, \Sigma_s \rangle \mapsto_p \Sigma_s(\text{GL_REG_0_ATI})$

5.2 Setting Constant Registers

$\langle \text{glSetFragmentShaderATI}(\text{GL_CON_x_ATI}, v); \text{ConstSet}_1 \dots \text{ConstSet}_{(n_c-1)}, \Sigma_s \rangle \mapsto_{cs}$
 $\langle \text{ConstSet}_0 \dots \text{ConstSet}_{(n_c-1)}, \Sigma_s[\text{GL_CON_x_ATI} \leftarrow c] \rangle$
 where $c = (v[0], v[1], v[2], v[3])$

5.3 Evaluation of a Pass

$\Sigma_{GL} \vdash \langle M_{s_0} \dots M_{s_{(n_s-1)}} M_{o_0} \dots M_{o_{(n_o-1)}} \text{Ps}_0 \dots \text{Ps}_{(n_p-1)}, \Sigma_s \rangle \mapsto_p$
 $\Sigma_{GL} \vdash \langle M_{o_0} \dots M_{o_{(n_o-1)}} \text{Ps}_0 \dots \text{Ps}_{(n_p-1)}, \Sigma'_s \rangle$
 where $\langle M_{s_0} \dots M_{s_{(n_s-1)}}, \Sigma_s \rangle \rightarrow_{ss} \langle \emptyset, \Sigma'_s \rangle$

$$\begin{array}{l} \Sigma_{GL} \vdash \langle M_{o_0} \dots M_{o_{(n_o-1)}} \text{Ps}_0 \dots \text{Ps}_{(n_p-1)}, \Sigma_s \rangle \quad \mapsto_p \\ \Sigma_{GL} \vdash \langle \text{Ps}_0 \dots \text{Ps}_{(n_p-1)}, \Sigma'_s \rangle \\ \text{where } \langle M_{o_0} \dots M_{o_{(n_o-1)}}, \Sigma_s \rangle \rightarrow_{op} \langle \emptyset, \Sigma'_s \rangle \end{array}$$

5.4 Sample Stage

$$\begin{array}{l} \Sigma_{GL} \vdash \langle \text{glPassTexCoordATI}(\text{GL_REG_}x_ATI, \text{Coord}, \text{Swizzle}); M_{s_1} \dots M_{s_{(n_s-1)}}, \Sigma_s \rangle \quad \mapsto_{ss} \\ \Sigma_{GL} \vdash \langle M_{s_1} \dots M_{s_{(n_s-1)}}, \Sigma_s[\text{REG_}x_ATI \leftarrow c] \rangle \\ \text{where Coord GetTuple } \Sigma_{GL}, \Sigma_s \rightarrow c' \\ c' \text{ DoSwizzle Swizzle } \rightarrow c \end{array}$$

$$\begin{array}{l} \Sigma_{GL} \vdash \langle \text{glSampleMapATI}(\text{GL_REG_}x_ATI, \text{Coord}, \text{Swizzle}); M_{s_1} \dots M_{s_{(n_s-1)}}, \Sigma_s \rangle \quad \mapsto_{ss} \\ \Sigma_{GL} \vdash \langle M_{s_1} \dots M_{s_{(n_s-1)}}, \Sigma_s[\text{REG_}x_ATI \leftarrow c] \rangle \\ \text{where Coord GetTuple } \Sigma_{GL}, \Sigma_s \rightarrow c' \\ c' \text{ DoSwizzle Swizzle } \rightarrow c \\ c'' \text{ DoSample } x, \Sigma_{GL} \rightarrow c \end{array}$$

5.5 Operation Stage

$$\begin{array}{l} \Sigma_{GL} \vdash \langle \text{glColorFragmentOp}x_ATI(\text{OpCode}, \text{GL_REG_}y_ATI, \text{DstMask}, \text{DstMod}, \text{SrcSpec}[x]); \\ M_{o_1} \dots M_{o_{(n_o-1)}}, \Sigma_s \rangle \quad \mapsto_{op} \\ \Sigma_{GL} \vdash \langle M_{o_1} \dots M_{o_{(n_o-1)}}, \Sigma_s[\text{GL_REG_}x_ATI \leftarrow c] \rangle \\ \text{where OpCode ApplyOp}_x \text{ SrcSpec}[x] \rightarrow c'' \\ \text{DstMod ApplyMod } c'' \rightarrow c' \\ \Sigma_s(\text{GL_REG_}y_ATI) \text{ ApplyMask } c \text{ with DstMask } \rightarrow c \end{array}$$

$$\begin{array}{l} \Sigma_{GL} \vdash \langle \text{glAlphaFragmentOp}x_ATI(\text{OpCode}, \text{GL_REG_}y_ATI, \text{DstMod}, \text{SrcSpec}[x]); \\ M_{o_1} \dots M_{o_{(n_o-1)}}, \Sigma_s \rangle \quad \mapsto_{op} \\ \Sigma_{GL} \vdash \langle M_{o_1} \dots M_{o_{(n_o-1)}}, \Sigma_s[\text{GL_REG_}x_ATI \leftarrow c] \rangle \\ \text{where OpCode ApplyOp}_x \text{ SrcSpec}[x]' \rightarrow c'' \\ \text{SrcSpec}[x] \text{ GetArgs } \Sigma_{GL}, \Sigma_s \rightarrow \text{SrcSpec}[x]' \\ \text{DstMod DoMod } c'' \rightarrow c' \\ \Sigma_s(\text{GL_REG_}y_ATI) \text{ DoAlpha } c \rightarrow c \end{array}$$

5.6 Helper Functions

5.6.1 GetTuple

$$\begin{array}{l} \text{GL_REG_}x_ATI \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad \Sigma_s(\text{REG_}x_ATI) \\ \text{GL_CON_}x_ATI \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad c \\ \text{if GL_CON_}x_ATI \in \Sigma_s \text{ then } c = \Sigma_s(\text{GL_CON_}x_ATI) \\ \text{else } c = \Sigma_{GL}(\text{GL_CON_}x_ATI) \\ \text{GL_TEXTURE}x_ARB \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad \Sigma_{GL}(\text{GL_TEXTURE}x_ARB) \\ \text{GL_ZERO} \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad (0, 0, 0, 0) \\ \text{GL_ONE} \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad (1, 1, 1, 1) \\ \text{GL_PRIMARY_COLOR_ARB} \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad \Sigma_{GL}(\text{GL_PRIMARY_COLOR_ARB}) \\ \text{GL_SECONDARY_INTERPOLATOR_ATI} \text{ GetTuple } \Sigma_{GL}, \Sigma_s \quad \mapsto \quad \Sigma_{GL}(\text{GL_SECONDARY_INTERPOLATOR_ATI}) \end{array}$$

5.6.2 Swizzle

(s, t, r, q) DoSwizzle GL_SWIZZLE_STR_ATI	\mapsto	$(s, t, r, \text{undefined})$
(s, t, r, q) DoSwizzle GL_SWIZZLE_STQ_ATI	\mapsto	$(s, t, q, \text{undefined})$
(s, t, r, q) DoSwizzle GL_SWIZZLE_STR_DQ_ATI	\mapsto	$(s/r, t/r, 1/r, \text{undefined})$
(s, t, r, q) DoSwizzle GL_SWIZZLE_STQ_DQ_ATI	\mapsto	$(s/q, t/q, 1/q, \text{undefined})$

5.6.3 DoSample

(s, t, r, q) DoSample GL_TEXTUREx_ARB, Σ_{GL}	\mapsto	c
--	-----------	-----

This is an OpenGL functionality that is not in the scope of this language.

It samples the texture $\Sigma_{GL}(\text{GL_TEXTUREx_ARB})$ at

the coordinates $(s, t, r, q), (s, t, r), (s, t)$ or (s) depending

on the actual dimensionality of the texture that is bound to

$\Sigma_{GL}(\text{GL_TEXTUREx_ARB})$.

5.6.4 DoMod

GL_NONE DoMod (s, t, r, q)	\mapsto	(s, t, r, q)
GL_2X_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(s * 2, t * 2, r * 2, q * 2)$
GL_4X_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(s * 4, t * 4, r * 4, q * 4)$
GL_8X_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(s * 8, t * 8, r * 8, q * 8)$
GL_HALF_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(s/2, t/2, r/2, q/2)$
GL_QUARTER_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(s/4, t/4, r/4, q/4)$
GL_EIGHTH_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(s/8, t/8, r/8, q/8)$
GL_SATURATE_BIT_ATI DoMod (s, t, r, q)	\mapsto	$(\text{clamp}(s), \text{clamp}(t), \text{clamp}(r), \text{clamp}(q))$

where $\text{clamp}(x) =$ if $x > 1$ then 1
else if $x < 0$ then 0
else x

5.6.5 ApplyMask

(s_1, t_1, r_1, q_1) ApplyMask (s_2, t_2, r_2, q_2) With NONE	\mapsto	(s_2, t_2, r_2, q_2)
(s_1, t_1, r_1, q_1) ApplyMask (s_2, t_2, r_2, q_2) With <i>bits</i>	\mapsto	c

where $\langle \text{bits}, (s_1, t_1, r_1, q_1) \rangle \rightarrow_m \langle \emptyset, c \rangle$

$\langle \text{GL_RED_BIT_ATI} \mid m_1 \mid \dots \mid m_n, (s_1, t_1, r_1, q_1) \rangle$	\mapsto	$\langle m_1 \mid \dots \mid m_n, (s_2, t_1, r_1, q_1) \rangle$
$\langle \text{GL_GREEN_BIT_ATI} \mid m_1 \mid \dots \mid m_n, (s_1, t_1, r_1, q_1) \rangle$	\mapsto	$\langle m_1 \mid \dots \mid m_n, (s_1, t_2, r_1, q_1) \rangle$
$\langle \text{GL_BLUE_BIT_ATI} \mid m_1 \mid \dots \mid m_n, (s_1, t_1, r_1, q_1) \rangle$	\mapsto	$\langle m_1 \mid \dots \mid m_n, (s_1, t_1, r_2, q_1) \rangle$

5.6.6 DoAlpha

(s_1, t_1, r_1, q_1) DoAlpha (s_2, t_2, r_2, q_2)	\mapsto	(s_1, t_1, r_1, q_2)
---	-----------	------------------------

5.6.7 GetArgs

a_0, \dots, a_n GetArgs Σ_{GL}, Σ_s	\mapsto	a'_0, \dots, a'_n
where a_0 GetModTuple Σ_{GL}, Σ_s	\mapsto	a'_0
\dots		
a_n GetModTuple Σ_{GL}, Σ_s	\mapsto	a'_n

Coord, SrcRep, SrcMod GetModTuple $\Sigma_{GL}, \Sigma_s \mapsto c$
 where Coord GetModTuple $\Sigma_{GL}, \Sigma_s \mapsto c''$
 c'' Replicate SrcRep $\mapsto c'$
 c' SourceMod SrcMod $\mapsto c$

5.6.8 Replicate

(s, t, r, q) Replicate GL_NONE $\mapsto (s, t, r, q)$
 (s, t, r, q) Replicate GL_RED $\mapsto (s, s, s, s)$
 (s, t, r, q) Replicate GL_GREEN $\mapsto (t, t, t, t)$
 (s, t, r, q) Replicate GL_BLUE $\mapsto (r, r, r, r)$
 (s, t, r, q) Replicate GL_ALPHA $\mapsto (q, q, q, q)$

5.6.9 SourceMod

(s, t, r, q) SourceMod GL_NONE $\mapsto (s, t, r, q)$
 (s, t, r, q) SourceMod $m_0 \dots m_n \mapsto c$
 where $\langle m_0 \dots m_n, (s, t, r, q) \rangle \rightarrow_{mod} \langle \emptyset, c \rangle$

$\langle m_0 \dots m_{i-1} \text{ GL_COMP_BIT_ATI } m_{i+1} \dots m_n, (s, t, r, q) \rangle \rightarrow_{mod}$
 $\langle m_0 \dots m_{i-1} m_{i+1} \dots m_n, (1-s, 1-t, 1-r, 1-q) \rangle$

$\langle m_0 \dots m_{i-1} \text{ GL_BIAS_BIT_ATI } m_{i+1} \dots m_n, (s, t, r, q) \rangle \rightarrow_{mod}$
 $\langle m_0 \dots m_{i-1} m_{i+1} \dots m_n, (s-0.5, t-0.5, r-0.5, q-0.5) \rangle$

$\langle m_0 \dots m_{i-1} \text{ GL_2X_BIT_ATI } m_{i+1} \dots m_n, (s, t, r, q) \rangle \rightarrow_{mod}$
 $\langle m_0 \dots m_{i-1} m_{i+1} \dots m_n, (s*2, t*2, r*2, q*2) \rangle$

$\langle m_0 \dots m_{i-1} \text{ GL_NEGATE_BIT_ATI } m_{i+1} \dots m_n, (s, t, r, q) \rangle \rightarrow_{mod}$
 $\langle m_0 \dots m_{i-1} m_{i+1} \dots m_n, (-s, -t, -r, -q) \rangle$

5.6.10 ApplyOp

OpCode ApplyOp args' $\mapsto c''$

GL_MOV_ATI ApplyOp₁ $(s_1, t_1, r_1, q_1) \mapsto (s_1, t_1, r_1, q_1)$
 GL_ADD_ATI ApplyOp₂ $(s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2) \mapsto (s_1 + s_2, t_1 + t_2, r_1 + r_2, q_1 + q_2)$
 GL_SUB_ATI ApplyOp₂ $(s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2) \mapsto (s_1 - s_2, t_1 - t_2, r_1 - r_2, q_1 - q_2)$
 GL_MUL_ATI ApplyOp₂ $(s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2) \mapsto (s_1 * s_2, t_1 * t_2, r_1 * r_2, q_1 * q_2)$
 GL_DOT3_ATI ApplyOp₂ $(s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2) \mapsto (c, c, c, c)$
 where $c = s_1 * s_2 + t_1 * t_2 + r_1 * r_2$

GL_DOT4_ATI ApplyOp₂ $(s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2) \mapsto (c, c, c, c)$
 where $c = s_1 * s_2 + t_1 * t_2 + r_1 * r_2 + q_1 * q_2$

GL_DOT2_ADD_ATI ApplyOp₃ (s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2)(s_3, t_3, r_3, q_3) \mapsto (c, c, c, c)
 where $c = s_1 * s_2 + t_1 * t_2 + r_3$

GL_MAD_ATI ApplyOp₃ (s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2)(s_3, t_3, r_3, q_3) \mapsto
 ($s_1 * s_2 + s_3$,
 $t_1 * t_2 + t_3$,
 $r_1 * r_2 + r_3$,
 $q_1 * q_2 + q_3$)

GL_LERP_ATI ApplyOp₃ (s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2)(s_3, t_3, r_3, q_3) \mapsto
 ($s_1 * s_2 + (1 - s_1) * s_3$,
 $t_1 * t_2 + (1 - t_1) * t_3$,
 $r_1 * r_2 + (1 - r_1) * r_3$,
 $q_1 * q_2 + (1 - q_1) * q_3$)

GL_CND_ATI ApplyOp₃ (s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2)(s_3, t_3, r_3, q_3) \mapsto ($new_s, new_t, new_r, new_q$)
 where if $s_3 > 0.5$ then $new_s = s_2$ else $new_s = s_3$
 if $t_3 > 0.5$ then $new_t = t_2$ else $new_t = t_3$
 if $r_3 > 0.5$ then $new_r = r_2$ else $new_r = r_3$
 if $q_3 > 0.5$ then $new_q = q_2$ else $new_q = q_3$

GL_CND0_ATI ApplyOp₃ (s_1, t_1, r_1, q_1)(s_2, t_2, r_2, q_2)(s_3, t_3, r_3, q_3) \mapsto ($new_s, new_t, new_r, new_q$)
 where if $s_3 > 0.0$ then $new_s = s_2$ else $new_s = s_3$
 if $t_3 > 0.0$ then $new_t = t_2$ else $new_t = t_3$
 if $r_3 > 0.0$ then $new_r = r_2$ else $new_r = r_3$
 if $q_3 > 0.0$ then $new_q = q_2$ else $new_q = q_3$