

COOLERSPACE Supplementary Materials

1 CoolerSpace Language Specification

Sec. 1.1 defines the complete syntax of COOLERSPACE. Sec. 1.2 defines permissible castings dictated by color science principles. Sec. 1.3 defines the formal typing rules.

1.1 CoolerSpace Syntax

Table 1: COOLERSPACE syntax.

ARRAYS	$a \in \text{floating point arrays}$
VARIABLE NAMES	$x \in \text{variable names}$
TRISTIMULUS COLOR TYPES	$\tau_{\text{tristimulus}} ::= \tau_{\text{XYZ}} \tau_{\text{LMS}} \tau_{\text{sRGB}} \tau_{\text{opRGB}}$
PERCEPTUAL COLOR TYPES	$\tau_{\text{perceptual}} ::= \tau_{\text{HSL}} \tau_{\text{LAB}}$
COLOR TYPES	$\tau_{\text{color}} ::= \tau_{\text{tristimulus}} \tau_{\text{perceptual}}$
SPECTRAL TYPES	$\tau_{\text{spectrum}} ::= \tau_{\text{Light}} \tau_{\text{Reflectance}} \tau_{\text{Scattering}} \tau_{\text{Absorption}} \tau_{\text{Pigment}}$
PHYSICAL TYPES	$\tau ::= \tau_{\text{color}} \tau_{\text{spectrum}} \tau_{\text{Chromaticity}} \tau_{\text{Matrix}}$
DIMENSION TYPES	$d ::= \mathbb{N} d \times d$
SHAPED TYPES	$s ::= (\tau, d)$
XYZ CHANNELS	$c_{\text{XYZ}} ::= X Y Z$
LMS CHANNELS	$c_{\text{LMS}} ::= L M S$
sRGB CHANNELS	$c_{\text{sRGB}} ::= R G B$
opRGB CHANNELS	$c_{\text{opRGB}} ::= R G B$
HSV CHANNELS	$c_{\text{HSL}} ::= H S V$
LAB CHANNELS	$c_{\text{LAB}} ::= L A B$
CHANNELS	$c ::= c_{\text{XYZ}} c_{\text{LMS}} c_{\text{sRGB}} c_{\text{opRGB}} c_{\text{HSL}} c_{\text{LAB}}$
BINARY OPERATORS	$\oplus ::= + - \times /$
VALUES	$v ::= x \tau(a) \tau(v) \tau(v, v) \text{mix}(v, v, v, v) v \oplus v \text{matmul}(v, v) v.c$
EXPRESSIONS	$e ::= x = v$
PROGRAMS	$P ::= e; P e$

1.2 Casting Graph

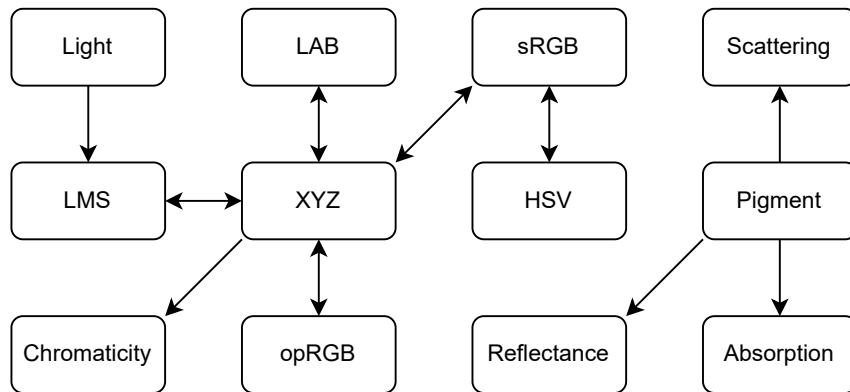


Figure 1: The graph showing all permissible castings.

1.3 CoolerSpace Type System

$$\begin{array}{c}
\frac{\tau \notin \{\tau_{\text{Pigment}}, \tau_{\text{Matrix}}\} \quad d_1 = d_2 \times \text{channel_count}(t) \quad a : d_1}{\Gamma \vdash \tau(a) : (t, d_2)} \text{INIT} \\
\\
\frac{a : d}{\Gamma \vdash \tau_{\text{Matrix}}(a) : (\tau_{\text{Matrix}}, d)} \text{MATRIXINIT} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Absorption}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{Scattering}}, d)}{\Gamma \vdash \tau_{\text{Pigment}}(v_1, v_2) : (\tau_{\text{Pigment}}, d)} \text{PIGMENTINIT} \\
\\
\frac{}{\text{broadcastable}(d, d)} \text{TRIVIALBROADCAST} \\
\\
\frac{}{\text{broadcastable}(1, d)} \text{SCALARBROADCAST} \\
\\
\frac{d_1 = d_2 \times d_3}{\text{broadcastable}(d_3, d_1)} \text{SUBSETBROADCAST} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Light}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{Light}}, d)}{\Gamma \vdash v_1 + v_2 : (\tau_{\text{Light}}, d)} \text{LIGHTADD} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{tristimulus}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{tristimulus}}, d)}{\Gamma \vdash v_1 + v_2 : (\tau_{\text{tristimulus}}, d)} \text{TRISTIMULUSADD} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{perceptual}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{perceptual}}, d)}{\Gamma \vdash v_1 + v_2 : (\tau_{\text{perceptual}}, d)} \text{PERCEPTUALADD} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, d_2) \quad \text{broadcastable}(d_2, d_1)}{\Gamma \vdash v_1 + v_2 : (\tau_{\text{Matrix}}, d_1)} \text{MATRIXADDL} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, d_2) \quad \text{broadcastable}(d_1, d_2)}{\Gamma \vdash v_1 + v_2 : (\tau_{\text{Matrix}}, d_2)} \text{MATRIXADDR} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Light}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{Reflectance}}, d)}{\Gamma \vdash v_1 \times v_2 : (\tau_{\text{Light}}, d)} \text{REFLECT} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{tristimulus}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, \text{channel_count}(\tau_{\text{tristimulus}}))}{\Gamma \vdash v_1 \times v_2 : (\tau_{\text{tristimulus}}, d)} \text{TRIChSCALE} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, d_2) \quad \text{broadcastable}(d_2, d_1)}{\Gamma \vdash v_1 \times v_2 : (\tau_{\text{Matrix}}, d_1)} \text{MATRIXMULL} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, d_2) \quad \text{broadcastable}(d_1, d_2)}{\Gamma \vdash v_1 \times v_2 : (\tau_{\text{Matrix}}, d_2)} \text{MATRIXMULR} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, d_2) \quad \text{broadcastable}(d_2, d_1)}{\Gamma \vdash v_1 / v_2 : (\tau_{\text{Matrix}}, d_1)} \text{MATRIXDIVL} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, d_2) \quad \text{broadcastable}(d_1, d_2)}{\Gamma \vdash v_1 / v_2 : (\tau_{\text{Matrix}}, d_2)} \text{MATRIXDIVR} \\
\\
\frac{\Gamma \vdash v_1 : (\tau_{\text{tristimulus}}, d) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, \text{channel_count}(\tau_{\text{tristimulus}}) \times \text{channel_count}(\tau_{\text{tristimulus}}))}{\Gamma \vdash \text{matmul}(v_1, v_2) : (\tau_{\text{tristimulus}}, d)} \text{TRIMATMUL}
\end{array}$$

$$\begin{array}{c}
\frac{\Gamma \vdash v_1 : (\tau_{\text{Matrix}}, d_1 \times \mathbb{N}) \quad \Gamma \vdash v_2 : (\tau_{\text{Matrix}}, \mathbb{N} \times d_2)}{\Gamma \vdash \text{matmul}(v_1, v_2) : (\tau_{\text{Matrix}}, d_1 \times d_2)} \text{ MATMUL} \\
\\
\frac{\Gamma \vdash v_1, v_2 : (\tau_{\text{Pigment}}, d) \quad \Gamma \vdash v_3, v_4 : (\tau_{\text{Matrix}}, d)}{\Gamma \vdash \text{mix}(v_3, v_1, v_4, v_2) : (\tau_{\text{Pigment}}, d)} \text{ PIGMENTMIX} \\
\\
\frac{\Gamma \vdash v : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\Gamma \vdash \tau_2(v) : (\tau_2, d)} \text{ CAST} \\
\\
\frac{\tau \neq \tau_{\text{Pigment}} \quad \Gamma \vdash v : (\tau, d)}{\Gamma \vdash \tau_{\text{Matrix}}(v) : (\tau_{\text{Matrix}}, d \times \text{channel_count}(\tau))} \text{ MATRIXCAST} \\
\\
\frac{\Gamma \vdash v : (\tau_i, d) \quad c \in c_i}{\Gamma \vdash v.c : (\tau_{\text{Matrix}}, d)} \text{ CHANNELGET}
\end{array}$$

Figure 2: COOLERSPACE typing rules. $\text{path_exists}(\tau_1, \tau_2)$ in the CAST rule type checks only if there exists a path from τ_1 to τ_2 in Fig. 1.

2 Translational Soundness

Programs written in COOLERSPACE are translated to ONNX [11]. This section proves that COOLERSPACE is translationally sound. For the translation between COOLERSPACE to ONNX to be sound, every well-typed value in COOLERSPACE must be translated to a well-typed value in ONNX. Translational soundness indicates that our translation preserves typeability: the type safety of COOLERSPACE is as strong as that of ONNX.

To succinctly demonstrate our proof strategy without losing generality, we have defined a subset of our language, dubbed SMALLERSPACE (Sec. 2.1). The syntax for SMALLERSPACE can be found in Tbl. 3, and the corresponding type rules can be found in Fig. 3. We have also formalized a subset of ONNX that is relevant to our translation (Sec. 2.2). The syntax can be found in Tbl. 4, and the corresponding type rules are in Fig. 4. Sec. 2.3 defines the formal translational semantics from SMALLERSPACE to ONNX. Sec. 2.4 shows the proof of translational soundness.

Tbl. 2 defines a channel count lookup table that will be referenced by the `channel_count(·)` function used during translation and type checking in this section.

<code>channel_count</code> (τ_{LMS})	= 3
<code>channel_count</code> (τ_{XYZ})	= 3
<code>channel_count</code> (τ_{Light})	= 89
<code>channel_count</code> ($\tau_{\text{Reflectance}}$)	= 89

Table 2: Channel count lookup table

2.1 SmallerSpace Syntax and Typing Rules

Natural numbers	$\mathbb{N} \in \text{natural numbers}$
Arrays	$a \in \text{floating point array literals}$
Dimension types	$d ::= \mathbb{N} d \times d$
Color types	$\tau_{\text{color}} ::= \tau_{\text{XYZ}} \tau_{\text{LMS}} \tau_{\text{sRGB}}$
Spectral types	$\tau_{\text{spectrum}} ::= \tau_{\text{Light}} \tau_{\text{Reflectance}}$
Physical types	$\tau ::= \tau_{\text{color}} \tau_{\text{spectrum}}$
Shaped types	$s ::= (\tau, d)$
Values	$v ::= \tau(a) \tau(v) v + v v \times v$

Table 3: SMALLERSPACE syntax

$\frac{d_1 = d_2 \times \text{channel_count}(\tau) \quad a : d_1}{\tau(a) : (\tau, d_2)} \text{INIT}$
$\frac{v : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\tau_2(v) : (\tau_2, d)} \text{CAST}$
$\frac{v_1, v_2 : (\tau_{\text{color}}, d)}{v_1 + v_2 : (\tau_{\text{color}}, d)} \text{COLORADD}$
$\frac{v_1, v_2 : (\tau_{\text{Light}}, d)}{v_1 + v_2 : (\tau_{\text{Light}}, d)} \text{LIGHTADD}$
$\frac{v_1 : (\tau_{\text{Light}}, d) \quad v_2 : (\tau_{\text{Reflectance}}, d)}{v_1 \times v_2 : (\tau_{\text{Light}}, d)} \text{REFLECT}$

Figure 3: SMALLERSPACE type rules.

2.2 ONNX Syntax and Typing Rules

NATURAL NUMBERS	$\mathbb{N} \in \text{natural numbers}$
ARRAYS	$a \in \text{floating point arrays}$
DIMENSION TYPES	$d ::= \mathbb{N} d \times d$
VALUES	$u ::= a \text{add}(u, u) \text{div}(u, u) \text{mul}(u, u) \text{sub}(u, u) \text{matmul}(u, u) \text{pow}(u, u)$

Table 4: Simplified ONNX syntax.

$\frac{}{\text{broadcastable}(d, d)}$	TRIVIALBROADCAST
$\frac{}{\text{broadcastable}(1, d)}$	SCALARBROADCAST
$\frac{d_1 = d_2 \times d_3}{\text{broadcastable}(d_3, d_1)}$	SUBSETBROADCAST
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_2, d_1)}{\text{add}(u_1, u_2) : d_1}$	ONNXADDL
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_1, d_2)}{\text{add}(u_1, u_2) : d_2}$	ONNXADDR
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_2, d_1)}{\text{sub}(u_1, u_2) : d_1}$	ONNXSUBL
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_1, d_2)}{\text{sub}(u_1, u_2) : d_2}$	ONNXSUBR
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_2, d_1)}{\text{mul}(u_1, u_2) : d_1}$	ONNXMULL
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_1, d_2)}{\text{mul}(u_1, u_2) : d_2}$	ONNXMULR
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_2, d_1)}{\text{div}(u_1, u_2) : d_1}$	ONNXDIVL
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_1, d_2)}{\text{div}(u_1, u_2) : d_2}$	ONNXDIVR
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_2, d_1)}{\text{pow}(u_1, u_2) : d_1}$	ONNXPowL
$\frac{u_1 : d_1 \quad u_2 : d_2 \quad \text{broadcastable}(d_1, d_2)}{\text{pow}(u_1, u_2) : d_2}$	ONNXPowR

Figure 4: Simplified ONNX type rules.

2.3 Translational Semantics

$\llbracket \mathbb{N} \rrbracket \triangleq \mathbb{N}$	T-Nat
$\llbracket v_1 + v_2 \rrbracket \triangleq \Phi(+, \tau_1, \tau_2)(v_1, v_2), \quad v_1 : (\tau_1, d_1), v_2 : (\tau_2, d_2)$	T-Add
$\llbracket a \rrbracket \triangleq a$	T-Array
$\llbracket v_1 \times v_2 \rrbracket \triangleq \Phi(\times, \tau_1, \tau_2)(v_1, v_2), \quad v_1 : (\tau_1, d_1), v_2 : (\tau_2, d_2)$	T-Mul
$\llbracket \tau(a) \rrbracket \triangleq a$	T-Init
$\llbracket \tau_d(v_o) \rrbracket \triangleq \Psi(\tau_o, \tau_d)(v_o), \quad v_o : (\tau_o, d_o)$	T-Cast
$\llbracket d \rrbracket \triangleq d$	T-Dim
$\llbracket (\tau, d) \rrbracket \triangleq d \times \text{channel_count}(\tau)$	T-Type
$\Phi(+, \tau_{XYZ}, \tau_{XYZ})(v_1, v_2) \triangleq \text{add}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)$	
$\Phi(+, \tau_{LMS}, \tau_{LMS})(v_1, v_2) \triangleq \text{add}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)$	
$\Phi(+, \tau_{sRGB}, \tau_{sRGB})(v_1, v_2) \triangleq \text{mul}(\text{pow}(\text{add}(\text{pow}(\text{div}(\llbracket v_1 \rrbracket, [255]), [2.2]), \text{div}(\text{pow}(\llbracket v_2 \rrbracket, [255]) [2.2])), [0.455]), [255])$	
$\Phi(\times, \tau_{\text{Light}}, \tau_{\text{Reflectance}})(v_1, v_2) \triangleq \text{mul}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)$	
$\Psi(\tau_{\text{Light}}, \tau_{\text{LMS}})(v) \triangleq \text{matmul}(\llbracket v \rrbracket, M_1)$	
$\Psi(\tau_{\text{LMS}}, \tau_{\text{XYZ}})(v) \triangleq \text{matmul}(\llbracket v \rrbracket, M_2)$	
$\Psi(\tau_{\text{XYZ}}, \tau_{\text{sRGB}})(v) \triangleq \text{pow}(\text{mul}(\text{matmul}(\llbracket v \rrbracket, M_3), [255]), [2.2])$	

Table 5: SMALLERSPACE to ONNX translational semantics. M_1 , M_2 , and M_3 represent the LMS Cone Fundamentals, XYZ to LMS transformation matrix, and XYZ to RGB transformation matrix, respectively. They are constant matrices that can be found in standard color science texts [15], and are omitted here.

2.4 Translational Soundness Proof

Theorem. The theorem for translational soundness is given below:

$$\frac{\llbracket v : (\tau, d) \rrbracket}{\llbracket v \rrbracket : \llbracket (\tau, d) \rrbracket}$$

We will prove the theorem through structural induction. The induction hypothesis can be generally stated as follows:

$$\frac{\llbracket v_i \subset v \rrbracket \quad \llbracket v_i : (\tau_i, d_i) \rrbracket \quad \llbracket v : (\tau, d) \rrbracket}{\llbracket v_i \rrbracket : \llbracket (\tau_i, d_i) \rrbracket} \text{INDHYP}$$

We will be using the converse of several typing rules in SMALLERSPACE. These rules will be labeled as CONV-RULENAME. In order for a converse rule to be valid, the set of expressions that are type checked by a rule must only be type check-able by said rule. We will show this assumption to be true for every case. We also use the SUBST rule to indicate substitution of equivalent values.

In this proof we iterate over every rule in SMALLERSPACE, and show that each rule is translationally sound.

Base case: Init

Values of the form $\tau(a)$ do not have value subterms. This makes it our base case for induction. There is only one type rule under which $\tau(a)$ type checks: INIT. Therefore CONV-INIT is valid. We also introduce a new rule T-ARRTYPE to assist with this proof. T-ARRTYPE ensures that the dimension type of an array literal is preserved after translation.

$$\frac{\llbracket a : d \rrbracket}{\llbracket a \rrbracket : \llbracket d \rrbracket} \text{ T-ARRTYPE}$$

Given $\llbracket \tau(a) : (\tau, d) \rrbracket$, we can conclude that the translated type of $\llbracket \tau(a) \rrbracket$ is $d \times \text{channel_count}(\tau)$.

$$\begin{array}{c}
\frac{}{\llbracket \tau(a) : (\tau, d) \rrbracket} \text{CONV-INIT} \\
\frac{}{\llbracket a : d \times \text{channel_count}(\tau) \rrbracket} \text{T-ARRTYPE} \quad \frac{}{\llbracket a \rrbracket \triangleq a} \text{T-ARRAY} \\
\frac{}{\llbracket a \rrbracket : \llbracket d \times \text{channel_count}(\tau) \rrbracket} \text{SUBST} \quad \frac{}{\llbracket \tau(a) \rrbracket \triangleq a} \text{T-INIT} \\
\frac{a : \llbracket d \times \text{channel_count}(\tau) \rrbracket}{\llbracket \tau(a) \rrbracket : \llbracket d \times \text{channel_count}(\tau) \rrbracket} \text{T-DIM} \\
\frac{\llbracket \tau(a) \rrbracket : \llbracket d \times \text{channel_count}(\tau) \rrbracket}{\llbracket \tau(a) \rrbracket : d \times \text{channel_count}(\tau)} \text{SUBST}
\end{array}$$

$\llbracket(\tau, d)\rrbracket$ also translates to $d \times \text{channel_count}(\tau)$. Therefore, we can conclude that $\llbracket\tau(a)\rrbracket : \llbracket(\tau, d)\rrbracket$. This satisfies the translational soundness theorem.

$$\frac{\frac{\llbracket \tau(a) \rrbracket : d \times \text{channel_count}(\tau)}{\llbracket \tau(a) \rrbracket : \llbracket \tau, d \rrbracket} \quad \frac{\llbracket \tau, d \rrbracket \triangleq d \times \text{channel_count}(\tau)}{\text{T-TYPE}}}{\text{SUBST}}$$

Case 1: ColorAdd

In order for CONV-COLORADD to be a valid rule, the set of expressions that are type checked by COLORADD must only be type check-able by COLORADD. In other words, $v_1 + v_2 : (\tau_{\text{color}}, d)$ must type check if and only if the premises for COLORADD are satisfied. We know this to be true, as there is no other rule that type checks addition between color types.

The proof below shows the validity of COLORADD for $\tau_{\text{XYZ}} + \tau_{\text{XYZ}}$. This proof must be repeated for LMS and sRGB.

$$\frac{\frac{\frac{\llbracket v_1 + v_2 : (\tau_{\text{XYZ}}, d) \rrbracket}{\llbracket v_1, v_2 : (\tau_{\text{XYZ}}, d) \rrbracket} \text{ CONV-COLORADD} \quad \llbracket v_1, v_2 \subset v_1 + v_2 \rrbracket}{\llbracket v_1, v_2 \rrbracket : \llbracket (\tau_{\text{XYZ}}, d) \rrbracket} \text{ INDHYP} \quad \frac{\text{broadcastable}(\llbracket (\tau_{\text{XYZ}}, d) \rrbracket, \llbracket (\tau_{\text{XYZ}}, d) \rrbracket)}{\text{ONNXADDR}} \quad \frac{\llbracket v_1, v_2 : (\tau_{\text{XYZ}}, d) \rrbracket}{\llbracket v_1 + v_2 \rrbracket \triangleq \text{add}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)} \text{ T-ADD}}{\llbracket v_1 + v_2 \rrbracket : \llbracket (\tau_{\text{XYZ}}, d) \rrbracket} \text{ SUBST}$$

Case 2: LightAdd

Like COLORADD, LIGHTADD is the only rule that can type check $v_1 + v_2 : (\tau_{\text{Light}}, d)$. Thus, CONV-LIGHTADD is valid. The proof tree is similar to that of COLORADD.

$$\infty \quad \frac{\frac{\frac{\llbracket v_1 + v_2 : (\tau_{\text{Light}}, d) \rrbracket}{\llbracket v_1, v_2 : (\tau_{\text{Light}}, d) \rrbracket} \text{ CONV-LIGHTADD} \quad \llbracket v_1, v_2 \subset v_1 + v_2 \rrbracket}{\llbracket v_1, v_2 \rrbracket : \llbracket (\tau_{\text{Light}}, d) \rrbracket} \text{ INDHYP} \quad \frac{\text{broadcastable}(\llbracket (\tau_{\text{Light}}, d) \rrbracket, \llbracket (\tau_{\text{Light}}, d) \rrbracket)}{\text{ONNXADDR}} \quad \frac{\llbracket v_1, v_2 : (\tau_{\text{Light}}, d) \rrbracket}{\llbracket v_1 + v_2 \rrbracket \triangleq \text{add}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)} \text{ T-ADD}}{\llbracket v_1 + v_2 \rrbracket : \llbracket (\tau_{\text{Light}}, d) \rrbracket} \text{ SUBST}$$

Case 3: Reflect

REFLECT is the only way by which $v_1 \times v_2$ is type checked in SMALLERSPACE. Therefore CONV-REFLECT is valid. Given $\llbracket v_1 \times v_2 : (\tau_{\text{Light}}, d) \rrbracket$, we can conclude that $\llbracket v_1 \rrbracket$ and $\llbracket v_2 \rrbracket$ are of type $d \times \text{channel_count}(\tau_{\text{Light}})$ after translation.

$$\frac{\frac{\frac{\llbracket v_1 \times v_2 : (\tau_{\text{Light}}, d) \rrbracket}{\llbracket v_1 : (\tau_{\text{Light}}, d) \rrbracket} \text{ CONV-REFLECT} \quad \llbracket v_1 \subset v_1 \times v_2 \rrbracket}{\llbracket v_1 \rrbracket : \llbracket (\tau_{\text{Light}}, d) \rrbracket} \text{ INDHYP} \quad \frac{\llbracket v_1 \rrbracket : \llbracket (\tau_{\text{Light}}, d) \rrbracket}{\llbracket v_1 \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}})} \text{ T-DIM}}{\frac{\frac{\llbracket v_1 \times v_2 : (\tau_{\text{Light}}, d) \rrbracket}{\llbracket v_2 : (\tau_{\text{Reflectance}}, d) \rrbracket} \text{ INV-REFLECT} \quad \llbracket v_2 \subset v_1 \times v_2 \rrbracket}{\llbracket v_2 \rrbracket : \llbracket (\tau_{\text{Reflectance}}, d) \rrbracket} \text{ INDHYP} \quad \frac{\llbracket v_2 \rrbracket : \llbracket (\tau_{\text{Reflectance}}, d) \rrbracket}{\llbracket v_2 \rrbracket : d \times \text{channel_count}(\tau_{\text{Reflectance}})} \text{ T-DIM} \quad \frac{\text{channel_count}(\tau_{\text{Reflectance}}) = \text{channel_count}(\tau_{\text{Light}})}{\llbracket v_2 \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}})} \text{ SUBST}}$$

With this information, we can show that $\text{mul}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) : d \times \text{channel_count}(\tau_{\text{Light}})$.

$$\frac{\llbracket v_1 \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}}) \quad \llbracket v_2 \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}}) \quad \text{broadcastable}(d \times \text{channel_count}(\tau_{\text{Light}}), d \times \text{channel_count}(\tau_{\text{Light}}))}{\text{mul}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) : d \times \text{channel_count}(\tau_{\text{Light}})} \begin{array}{l} \text{TRIVBROAD} \\ \text{ONNXMULR} \end{array}$$

Since $v_1 \times v_2$ is translated to $\text{mul}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)$ with type $d \times \text{channel_count}(\tau_{\text{Light}})$, and since $\llbracket (\tau_{\text{Light}}, d) \rrbracket$ is also typed $d \times \text{channel_count}(\tau_{\text{Light}})$ after translation, we can conclude that REFLECT is translationally sound.

$$\frac{\frac{\text{mul}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) : d \times \text{channel_count}(\tau_{\text{Light}}) \quad \frac{\llbracket v_1 : (\tau_{\text{Light}}, d) \rrbracket \quad \llbracket v_2 : (\tau_{\text{Reflectance}}, d) \rrbracket}{\llbracket v_1 \times v_2 \rrbracket \triangleq \text{mul}(\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket)} \text{T-Mul}}{\llbracket v_1 \times v_2 \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}})} \text{SUBST} \quad \frac{\llbracket (\tau_{\text{Light}}, d) \rrbracket \triangleq d \times \text{channel_count}(\tau_{\text{Light}})}{\llbracket v_1 \times v_2 \rrbracket : \llbracket (\tau_{\text{Light}}, d) \rrbracket} \begin{array}{l} \text{T-TYPE} \\ \text{SUBST} \end{array}$$

Case 4: Cast

For any output physical type, there are multiple origin physical types that can satisfy the CAST rule. Therefore, we must show that CAST is translationally sound for every pair of types τ_1, τ_2 such that $\text{path_exists}(\tau_1, \tau_2)$. All these proofs are similar in structure. I will be showing the proof for Light to LMS.

We first show that $\llbracket v \rrbracket$ is of type $d \times \text{channel_count}(\tau_{\text{Light}})$ after translation.

$$\frac{\frac{\frac{\llbracket \tau_{\text{LMS}}(v) : (\tau_{\text{LMS}}, d) \rrbracket}{\llbracket v : (\tau_{\text{Light}}, d) \rrbracket} \text{CONV-CAST-LIGHT2LMS} \quad \llbracket v \subset \tau_{\text{LMS}}(v) \rrbracket}{\llbracket v \rrbracket : \llbracket (\tau_{\text{Light}}, d) \rrbracket} \text{INDHYP} \quad \frac{\llbracket (\tau_{\text{Light}}, d) \rrbracket \triangleq d \times \text{channel_count}(\tau_{\text{Light}})}{\llbracket v \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}})} \begin{array}{l} \text{T-TYPE} \\ \text{SUBST} \end{array}$$

$\llbracket \tau_{\text{LMS}}(v) \rrbracket$ translates to $\text{matmul}(\llbracket v \rrbracket, M_1)$, where M_1 is a constant matrix of dimension $\text{channel_count}(\tau_{\text{Light}}) \times \text{channel_count}(\tau_{\text{LMS}})$.

$$\frac{\llbracket v : (\tau_{\text{Light}}, d) \rrbracket}{\llbracket \tau_{\text{LMS}}(v) \rrbracket \triangleq \text{matmul}(\llbracket v \rrbracket, M_1)} \text{T-CAST}$$

We can conclude that $\llbracket \tau_{\text{LMS}}(v) \rrbracket : d \times \text{channel_count}(\tau_{\text{LMS}})$.

$$\frac{\frac{\llbracket v \rrbracket : d \times \text{channel_count}(\tau_{\text{Light}}) \quad M_1 : \text{channel_count}(\tau_{\text{Light}}) \times \text{channel_count}(\tau_{\text{LMS}})}{\text{matmul}(\llbracket v \rrbracket, M_1) : d \times \text{channel_count}(\tau_{\text{LMS}})} \text{ONNXMATMUL} \quad \frac{\llbracket \tau_{\text{LMS}}(v) \rrbracket \triangleq \text{matmul}(\llbracket v \rrbracket, M_1)}{\llbracket \tau_{\text{LMS}}(v) \rrbracket : d \times \text{channel_count}(\tau_{\text{LMS}})} \text{SUBST}$$

Since $\llbracket (\tau_{\text{LMS}}, d) \rrbracket$ also translates to $d \times \text{channel_count}(\tau_{\text{LMS}})$, the translational soundness theorem holds true for the CAST rule when the origin type is τ_{Light} and the destination type is τ_{LMS} .

$$\frac{\llbracket \tau_{\text{LMS}}(v) \rrbracket : d \times \text{channel_count}(\tau_{\text{LMS}}) \quad \frac{}{\llbracket (\tau_{\text{LMS}}, d) \rrbracket \triangleq d \times \text{channel_count}(\tau_{\text{LMS}})} \text{T-TYPE}}{\llbracket \tau_{\text{LMS}}(v) \rrbracket : \llbracket (\tau_{\text{LMS}}, d) \rrbracket} \text{SUBST}$$

Translational soundness for CAST must be proven for every valid pair of origin and destination types. The remaining CAST proofs are of similar form to the one above.

Conclusion

We have shown that every rule in SMALLERSPACE is translationally sound under the inductive hypothesis. We have also proven that a base case, INIT, is translationally sound independent of the inductive hypothesis. Therefore, by the principle of structural induction, SMALLERSPACE must be translationally sound.

3 Type Soundness Background Information

We can additionally show that COOLERSPACE is type sound. In this section we will describe the requisite information for the type soundness proof. Similar to our proof of translational soundness, We will prove type soundness over SMALLERSPACE, a subset of COOLERSPACE. We define SMALLERSPACE's syntax and type rules in Sec. 3.1. Evaluation rules are specified in Sec. 3.2. These evaluation rules represent the expected computation for each operation. In Sec. 4, we will utilize the information outlined in this section to prove that SMALLERSPACE is type sound.

3.1 SmallerSpace Syntax and Rules

The SMALLERSPACE syntax that we use here has been augmented with array expressions. These array expressions allow us to perform arithmetic on raw floating point arrays, akin to NumPy array operations. Like ONNX, broadcasting rules are also used in array expressions. The syntax for SMALLERSPACE can be found in Tbl. 6, and the corresponding type rules can be found in Fig. 5.

Natural numbers	$\mathbb{N} \in \text{natural numbers}$
Binary operators	$\oplus ::= + - \times / **$
Arrays	$a \in \mathbb{A}$ (set of floating point array literals)
Array Expressions	$\alpha ::= a (\alpha) \oplus (\alpha) \text{matmul}(\alpha, \alpha)$
Dimension types	$d ::= \mathbb{N} d \times d$
Tristimulus types	$\tau_{\text{tristimulus}} ::= \tau_{\text{XYZ}} \tau_{\text{LMS}} \tau_{\text{sRGB}}$
Spectral types	$\tau_{\text{spectrum}} ::= \tau_{\text{Light}} \tau_{\text{Reflectance}}$
Physical types	$\tau ::= \tau_{\text{tristimulus}} \tau_{\text{spectrum}}$
Shaped types	$s ::= (\tau, d)$
Values	$v ::= \tau(\alpha) \tau(v) v \oplus v$

Table 6: Augmented SMALLERSPACE syntax

$$\begin{array}{c}
\frac{\alpha : d \times \text{channel_count}(\tau)}{\tau(\alpha) : (\tau, d)} \text{INIT} \\
\\
\frac{v : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\tau_2(v) : (\tau_2, d)} \text{CAST} \\
\\
\frac{v_1, v_2 : (\tau_{\text{tristimulus}}, d)}{v_1 + v_2 : (\tau_{\text{tristimulus}}, d)} \text{TRISTIMULUSADD} \\
\\
\frac{v_1, v_2 : (\tau_{\text{Light}}, d)}{v_1 + v_2 : (\tau_{\text{Light}}, d)} \text{LIGHTADD} \\
\\
\frac{v_1 : (\tau_{\text{Light}}, d) \quad v_2 : (\tau_{\text{Reflectance}}, d)}{v_1 \times v_2 : (\tau_{\text{Light}}, d)} \text{REFLECT} \\
\\
\frac{}{\text{broadcastable}(d, d)} \text{TRIVIALBROADCAST} \\
\\
\frac{}{\text{broadcastable}(1, d)} \text{SCALARBROADCAST} \\
\\
\frac{d_1 = d_2 \times d_3}{\text{broadcastable}(d_3, d_1)} \text{SUBSETBROADCAST} \\
\\
\frac{\alpha_1 : d_1 \quad \alpha_2 : d_2 \quad \text{broadcastable}(d_2, d_1)}{(\alpha_1) \oplus (\alpha_2) : d_1} \text{ARRAYBINOPL} \\
\\
\frac{\alpha_1 : d_1 \quad \alpha_2 : d_2 \quad \text{broadcastable}(d_1, d_2)}{(\alpha_1) \oplus (\alpha_2) : d_2} \text{ARRAYBINOPR} \\
\\
\frac{\alpha_1 : d_1 \times d_i \quad \alpha_2 : d_i \times d_2}{\text{matmul}(\alpha_1, \alpha_2) : d_1 \times d_2} \text{ARRAYMATMUL}
\end{array}$$

Figure 5: Augmented SMALLERSPACE type rules.

3.2 Evaluation Rules

We have defined rules for evaluating SMALLERSPACE expressions. These rules can be found in Fig. 6. These evaluation rules operate on “**basic values**” like $\tau_{\text{Light}}(a_1)$ and $\tau_{\text{XYZ}}(a_2)$. a_1 and a_2 are array literals. Basic values are significant, as they represent the most reduced version of a colorspace value. They are defined only by a physical type and an array literal. As such, they cannot be evaluated any further. We will utilize v_B to indicate the set of all basic values.

$$\begin{aligned}
\tau_{\text{Light}}(a_1) + \tau_{\text{Light}}(a_2) &\rightarrow \tau_{\text{Light}}((a_1) + (a_2)) & \text{E-LIGHTADD} \\
\tau_{\text{XYZ}}(a_1) + \tau_{\text{XYZ}}(a_2) &\rightarrow \tau_{\text{XYZ}}((a_1) + (a_2)) & \text{E-XYZADD} \\
\tau_{\text{LMS}}(a_1) + \tau_{\text{LMS}}(a_2) &\rightarrow \tau_{\text{LMS}}((a_1) + (a_2)) & \text{E-LMSADD} \\
\tau_{\text{sRGB}}(\tau_{\text{sRGB}}(a_1) + \tau_{\text{sRGB}}(a_2) &\rightarrow \tau_{\text{sRGB}}(((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}}) + ((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1}) \times ([255])) & \text{E-sRGBADD} \\
\tau_{\text{Light}}(a_1) \times \tau_{\text{Reflectance}}(a_2) &\rightarrow \tau_{\text{Light}}((a_1) \times (a_2)) & \text{E-REFLECT}
\end{aligned}$$

Figure 6: Evaluation rule for SMALLERSPACE type rules. $\gamma_{\tau_{\text{sRGB}}}$ and $\gamma_{\tau_{\text{sRGB}}}^{-1}$ are 1 dimension type array constants that represent the gamma value of the sRGB color space. They are 2.2 and 2.2^{-1} , respectively.

We have several rules for evaluating castings. The rules listed in Fig. 7 represent edges in the casting graph (Fig. 1). In other words, they evaluate castings between adjacent types.

$$\begin{aligned}
\tau_{\text{LMS}}(\tau_{\text{Light}}(a)) &\rightarrow \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{LIGHT2LMS}})) & \text{E-LIGHT2LMS} \\
\tau_{\text{LMS}}(\tau_{\text{XYZ}}(a)) &\rightarrow \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{XYZ2LMS}})) & \text{E-XYZ2LMS} \\
\tau_{\text{XYZ}}(\tau_{\text{LMS}}(a)) &\rightarrow \tau_{\text{XYZ}}(\text{matmul}(a, M_{\text{LMS2XYZ}})) & \text{E-LMS2XYZ} \\
\tau_{\text{sRGB}}(\tau_{\text{XYZ}}(a)) &\rightarrow \tau_{\text{sRGB}}(\text{matmul}(a, M_{\text{XYZ2sRGB}}) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) & \text{E-XYZ2sRGB} \\
\tau_{\text{XYZ}}(\tau_{\text{sRGB}}(a)) &\rightarrow \tau_{\text{XYZ}}(\text{matmul}((a) * (\gamma_{\tau_{\text{sRGB}}}), M_{\text{sRGB2XYZ}})) & \text{E-sRGB2XYZ}
\end{aligned}$$

Figure 7: Adjacent casting rules. The M values represent array literal constants. All M values with the exception of M_{XYZ2LMS} are of dimension 3×3 . M_{XYZ2LMS} is of dimension 89×3

However, SMALLERSPACE also allows casting between types that are not adjacent on the casting graph. In order to achieve this, we have defined an additional function: $\text{distance}(\tau_1, \tau_2)$ returns the number of edges in the shortest path from τ_1 to τ_2 . We also define E-CASTHOP to evaluate casts between non-adjacent types.

$$\frac{\text{path_exists}(\tau_o, \tau_d) \quad 1 \leq \text{distance}(\tau_i, \tau_d) < \text{distance}(\tau_o, \tau_d) \quad \text{distance}(\tau_o, \tau_i) = 1}{\tau_d(\tau_o(a)) \rightarrow \tau_d(\tau_i(\tau_o(a)))} \text{E-CASTHOP}$$

We also need to account for castings from a type to itself. We design the E-CASTTRIVIAL evaluation relation for this purpose.

$$\tau(\tau(v)) \rightarrow \tau(v) \quad \text{E-CASTTRIVIAL}$$

E-PROPBINOPL and E-PROPBINOPR allow an evaluation step to be taken on a binary operation if either operand can be further evaluated. E-PROPCAST does the same for casts. E-PROPINIT fulfills a similar function for nested array expressions that can be further evaluated.

$$\begin{aligned}
\frac{v_1 \rightarrow v'_1}{v_1 \oplus v_2 \rightarrow v'_1 \oplus v_2} & \text{E-PROPBINOPL} \\
\frac{v_2 \rightarrow v'_2}{v_1 \oplus v_2 \rightarrow v_1 \oplus v'_2} & \text{E-PROPBINOPR}
\end{aligned}$$

$$\frac{v \rightarrow v'}{\tau(v) \rightarrow \tau(v')} \text{ E-PROPCAST}$$

$$\frac{\alpha \rightarrow \alpha'}{\tau(\alpha) \rightarrow \tau(\alpha')} \text{ E-PROPINIT}$$

Since we have augmented SMALLERSPACE with array expressions, we also need rules to evaluate those, namely E-ARRBINOPL, E-ARRBINOPR, and E-ARRMATMUL. These evaluation rules do not manipulate array literals, as array literals are abstracted in SMALLERSPACE. Instead, these evaluation rules utilize dimension type information to check if the evaluation is possible.

$$\frac{\alpha_1 : d_1 \quad \alpha_2 : d_2 \quad \alpha_1, \alpha_2 \in \mathbb{A} \quad \text{broadcastable}(d_2, d_1)}{\exists a_3 : d_1 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow a_3} \text{ E-ARRBINOPL}$$

$$\frac{\alpha_1 : d_1 \quad \alpha_2 : d_2 \quad \alpha_1, \alpha_2 \in \mathbb{A} \quad \text{broadcastable}(d_1, d_2)}{\exists a_3 : d_2 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow a_3} \text{ E-ARRBINOPR}$$

$$\frac{\alpha_1 : d_1 \times d_m \quad \alpha_2 : d_m \times d_2 \quad \alpha_1, \alpha_2 \in \mathbb{A}}{\exists a_3 : d_1 \times d_2 \mid \text{matmul}(\alpha_1, \alpha_2) \rightarrow a_3} \text{ E-ARRMATMUL}$$

We also need evaluation step propagation rules for array expressions as well.

$$\frac{\alpha_1 \rightarrow \alpha'_1}{(\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha'_1) \oplus (\alpha_2)} \text{ E-PROPARRBINOPL}$$

$$\frac{\alpha_2 \rightarrow \alpha'_2}{(\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha_1) \oplus (\alpha'_2)} \text{ E-PROPARRBINOPR}$$

$$\frac{\alpha_1 \rightarrow \alpha'_1}{\text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha'_1, \alpha_2)} \text{ E-PROPARRMATMULL}$$

$$\frac{\alpha_2 \rightarrow \alpha'_2}{\text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha_1, \alpha'_2)} \text{ E-PROPARRMATMULR}$$

4 Type Soundness Proof

We will prove type soundness through individual proofs of progress and preservation. A “basic value” is a value of the form $\tau(a)$. Basic values are significant as they are composed of only a physical type and an array literal. They cannot be evaluated any further. We will use v_B to indicate the set of all basic values.

We prove SMALLERSPACE is type sound by showing that SMALLERSPACE satisfies both progress and preservation. Our proof for progress can be found in Sec. 4.3. Our preservation proof is in Sec. 4.4. Since SMALLERSPACE satisfies both progress and preservation, SMALLERSPACE is type sound. In order to show that SMALLERSPACE is type sound, we first define a set of “converse rules” in Sec. 4.1. Then, we prove that SMALLERSPACE’s new array expressions also satisfy type soundness in Sec. 4.2.

4.1 Converse Rules

Similar to our proof of translational soundness, we will also be utilizing converse rules to prove type soundness. These rules will be labeled as CONV-RULENAME. In this section, we will detail each converse rule we utilize in our proof, and explain why they are valid.

$$\frac{\tau(\alpha) : (\tau, d)}{\alpha : d \times \text{channel_count}(\tau)} \text{ CONV-INIT}$$

The CONV-INIT type rule is the converse rule for the INIT type rule. CONV-INIT is valid because the INIT rule is the only rule that type checks values of the form $\tau(\alpha)$.

$$\frac{v : (\tau, d) \quad v \in v_B}{\exists a : d \times \text{channel_count}(\tau) \mid v = \tau(a)} \text{ CONV-BASICINIT}$$

CONV-BASICINIT represents an alternative converse rule for the INIT type rule. CONV-BASICINIT is valid as if v is a basic type, we know v must take the form of $\tau(a)$, where a is an array literal. Therefore, we can apply CONV-INIT to derive the dimension type of a .

$$\frac{v_1 + v_2 : (\tau_{\text{tristimulus}}, d)}{v_1, v_2 : (\tau_{\text{tristimulus}}, d)} \text{ CONV-TRISTIMULUSADD}$$

$$\frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_1, v_2 : (\tau_{\text{Light}}, d)} \text{ CONV-LIGHTADD}$$

$$\frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_1 : (\tau_{\text{Light}}, d) \wedge v_2 : (\tau_{\text{Light}}, d)} \text{ CONV-REFLECT}$$

$$\frac{\tau_2(v) : (\tau_2, d)}{\exists \tau_1 \mid v : (\tau_1, d) \wedge \text{path_exists}(\tau_1, \tau_2)} \text{ CONV-CAST}$$

$$\frac{\text{matmul}(\alpha_1, \alpha_2) : d}{\exists d_1, d_2, d_i \mid d = d_1 \times d_2 \wedge \alpha_1 : d_1 \times d_i \wedge \alpha_2 : d_i \times d_2} \text{ CONV-ARRAYMATMUL}$$

CONV-TRISTIMULUSADD, CONV-LIGHTADD, CONV-REFLECT, CONV-CAST, and CONV-ARRAYMATMUL are inverted rules corresponding to the TRISTIMULUSADD, LIGHTADD, REFLECT, CAST, and ARRAYMATMUL rules. These converse rules are valid as TRISTIMULUSADD, LIGHTADD, REFLECT, CAST, and ARRAYMATMUL are the only rules that are able to type check $v_1 + v_2 : (\tau_{\text{tristimulus}}, d)$, $v_1 + v_2 : (\tau_{\text{Light}}, d)$, $v_1 \times v_2 : (\tau_{\text{Light}}, d)$, $\tau_2(v) : (\tau_2, d)$, and $\text{matmul}(\alpha_1, \alpha_2) : d_1 \times d_2$.

$$\frac{(\alpha_1) \oplus (\alpha_2) : d}{(\exists d_1 \mid \alpha_1 : d_1 \wedge \alpha_2 : d \wedge \text{broadcastable}(d_1, d)) \vee (\exists d_2 \mid \alpha_2 : d_2 \wedge \alpha_1 : d \wedge \text{broadcastable}(d_2, d))} \text{ CONV-ARRBINOP}$$

$$\frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_1 \mid \alpha_1 : d_1 \wedge \alpha_2 : d \wedge \text{broadcastable}(d_1, d)} \text{ CONV-ARRBINOPL}$$

$$\frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_2 \mid \alpha_2 : d_2 \wedge \alpha_1 : d \wedge \text{broadcastable}(d_2, d)} \text{ CONV-ARRBINOPR}$$

Since there are two rules that can type check the expression $(\alpha_1) \oplus (\alpha_2)$ (ARRAYBINOPL and ARRAYBINOPR), there are two disjoint conclusions that can be drawn from $(\alpha_1) \oplus (\alpha_2) : d$. At least one of the conclusions must be true. We split CONV-ARRBINOP into two individual rules for ease of use: CONV-ARRBINOPL and CONV-ARRBINOPR. These two rules are **not valid in isolation**. Every proof case that uses the CONV-ARRBINOPL type rule must also show that their conclusion holds when the CONV-ARRBINOPR type rule is used instead (and vice versa).

4.2 Array Expressions Type Soundness

Before we can prove the type soundness of SMALLERSPACE, we must first prove that array expressions are type sound. In order to do this, we must show that array expressions satisfy both progress and preservation.

4.2.1 Array Expressions Progress

Theorem. A typed array expression α is either an array literal ($\alpha \in \mathbb{A}$) or can take an evaluation step to some α' . The theorem is formally stated below:

$$\frac{\alpha \notin \mathbb{A} \quad \alpha : d}{\exists \alpha' \mid \alpha \rightarrow \alpha'} \text{ARRAY-PROGRESS}$$

The progress theorem is paired with the following inductive hypothesis:

$$\frac{\alpha_i \subset \alpha \quad \alpha_i \notin \mathbb{A} \quad \alpha : d}{\exists \alpha'_i \mid \alpha_i \rightarrow \alpha'_i} \text{ARRAY-PROGHYP}$$

We will demonstrate that array expressions satisfy the progress theorem through proof by induction.

Case 1: $\alpha \in \mathbb{A}$. If $\alpha \in \mathbb{A}$, α is an array literal already and does not need to be further evaluated. The progress theorem holds for case 1.

Case 2: $\alpha = (\alpha_1) \oplus (\alpha_2)$ and $\alpha_1 \notin \mathbb{A}$.

$$\frac{\frac{\alpha_1 \subset (\alpha_1) \oplus (\alpha_2) \quad \alpha_1 \notin \mathbb{A} \quad (\alpha_1) \oplus (\alpha_2) : d}{\exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1} \text{ARRAY-PROGHYP}}{\exists \alpha'_1 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha'_1) \oplus (\alpha_2)} \text{E-PROPARRBINOPL}$$

Since $(\alpha_1) \oplus (\alpha_2)$ can be further evaluated to $(\alpha'_1) \oplus (\alpha_2)$, progress holds for this case.

Case 3: $\alpha = (\alpha_1) \oplus (\alpha_2)$ and $\alpha_2 \notin \mathbb{A}$. Similar to case 2.

$$\frac{\frac{\alpha_1 \subset (\alpha_1) \oplus (\alpha_2) \quad \alpha_2 \notin \mathbb{A} \quad (\alpha_1) \oplus (\alpha_2) : d}{\exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2} \text{ARRAY-PROGHYP}}{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha_1) \oplus (\alpha'_2)} \text{E-PROPARRBINOPR}$$

Since $(\alpha_1) \oplus (\alpha_2)$ can be further evaluated to $(\alpha_1) \oplus (\alpha'_2)$, progress holds for this case.

Case 4: $\alpha = (\alpha_1) \oplus (\alpha_2)$ and $\alpha_1, \alpha_2 \in \mathbb{A}$.

$$\frac{\frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_1 \mid \alpha_1 : d_1 \wedge \alpha_2 : d \wedge \text{broadcastable}(d_1, d)} \text{CONV-ARRBINOPL} \quad \alpha_1, \alpha_2 \in \mathbb{A}}{\exists a_3 : d \mid (\alpha_1) \oplus (\alpha_2) \rightarrow a_3} \text{E-ARRBINOPR}$$

Since CONV-ARRBINOPL is used in the above proof, we must also show that the conclusion holds when CONV-ARRBINOPR is used instead.

$$\frac{\frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_2 \mid \alpha_2 : d_2 \wedge \alpha_1 : d \wedge \text{broadcastable}(d_2, d)} \text{CONV-ARRBINOPR} \quad \alpha_1, \alpha_2 \in \mathbb{A}}{\exists a_3 : d \mid (\alpha_1) \oplus (\alpha_2) \rightarrow a_3} \text{E-ARRBINOPL}$$

Regardless of whether CONV-ARRBINOPL or CONV-ARRBINOPR is used, we have shown that $(\alpha_1) \oplus (\alpha_2)$ evaluates to some $a_3 : d$ in this case. Therefore, progress holds for this case.

Case 5: $\alpha = \text{matmul}(\alpha_1, \alpha_2)$ and $\alpha_1 \notin \mathbb{A}$.

$$\frac{\frac{\alpha_1 \subset \text{matmul}(\alpha_1, \alpha_2) \quad \alpha_1 \notin \mathbb{A} \quad \text{matmul}(\alpha_1, \alpha_2) : d}{\exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1} \text{ARRAY-PROGHYP}}{\exists \alpha'_1 \mid \text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha'_1, \alpha_2)} \text{E-PROPARRMATMULL}$$

Since $\text{matmul}(\alpha_1, \alpha_2)$ evaluates to $\text{matmul}(\alpha'_1, \alpha_2)$, progress holds for this case.

Case 6: $\alpha = \text{matmul}(\alpha_1, \alpha_2)$ and $\alpha_2 \notin \mathbb{A}$.

$$\frac{\frac{\alpha_2 \subset \text{matmul}(\alpha_1, \alpha_2) \quad \alpha_2 \notin \mathbb{A} \quad \text{matmul}(\alpha_1, \alpha_2) : d}{\exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2} \text{ARRAY-PROGHYP}}{\exists \alpha'_2 \mid \text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha_1, \alpha'_2)} \text{E-PROPARRMATMULR}$$

Since $\text{matmul}(\alpha_1, \alpha_2)$ evaluates to $\text{matmul}(\alpha_1, \alpha'_2)$, progress holds for this case.

Case 7: $\alpha = \text{matmul}(\alpha_1, \alpha_2)$ and $\alpha_1, \alpha_2 \in \mathbb{A}$.

$$\frac{\frac{\text{matmul}(\alpha_1, \alpha_2) : d}{\exists d_1, d_2, d_i \mid d = d_1 \times d_2 \wedge \alpha_1 : d_1 \times d_i \wedge \alpha_2 : d_i \times d_2} \text{CONV-ARRAYMATMUL} \quad \alpha_1, \alpha_2 \in \mathbb{A}}{\exists a_3 : d_1 \times d_2 \mid \text{matmul}(\alpha_1, \alpha_2) \rightarrow a_3} \text{E-ARRMATMUL}$$

Since $\text{matmul}(\alpha_1, \alpha_2)$ evaluates to some $a_3 : d_1 \times d_2$, progress holds for this case.

Summary. The seven cases that we prove progress for are comprehensive. In each case, we show that the corresponding array expression can take an evaluation step given the inductive hypothesis. Therefore, we have shown that the progress theorem holds for array expressions.

4.2.2 Array Expression Preservation

Theorem. A typed array expression α preserves its type after taking an evaluation step to α' . The theorem is formally stated below:

$$\frac{\alpha : d \quad \alpha \rightarrow \alpha'}{\alpha' : d} \text{ARRAY-PRESERVATION}$$

The preservation theorem is paired with the following inductive hypothesis:

$$\frac{\alpha_i \subset \alpha \quad \alpha_i \rightarrow \alpha'_i \quad \alpha_i : d}{\alpha'_i : d} \text{ARRAY-PRESHYP}$$

We will demonstrate that array expressions satisfy the preservation theorem through proof by structural induction. We will utilize the same cases that we utilized in the progress proof for array expressions.

Case 1: $\alpha \in \mathbb{A}$. If $\alpha \in \mathbb{A}$, α is an array literal already and cannot be evaluated further. Therefore, the preservation theorem does not apply.

Case 2: $\alpha = (\alpha_1) \oplus (\alpha_2)$ and $\alpha_1 \notin \mathbb{A}$.

$$\frac{\frac{\frac{\alpha_1 \notin \mathbb{A} \quad \frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_1 \mid \alpha_1 : d_1 \wedge \alpha_2 : d \wedge \text{broadcastable}(d_1, d)} \text{CONV-ARRBINOPL}}{\exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1} \text{ARRAY-PROGRESS}}{\exists \alpha'_1 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha'_1) \oplus (\alpha_2)} \text{E-PROPARRBINOPL}}{\frac{\alpha_2 : d \quad \exists d_1 \mid \text{broadcastable}(d_1, d) \quad \frac{\alpha_1 \subset (\alpha_1) \oplus (\alpha_2) \quad \exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1 \quad \exists d_1 \mid \alpha_1 : d_1}{\exists \alpha'_1, d_1 \mid \alpha'_1 : d_1} \text{ARRAY-PRESHYP}}{\exists \alpha'_1 \mid (\alpha'_1) \oplus (\alpha_2) : d} \text{ARRAYBINOPR}}$$

Since CONV-ARRBINOP_L is used in the above proof, we must also show that the conclusion holds when CONV-ARRBINOP_R is used instead.

$$\begin{array}{c}
\frac{\alpha_1 \notin \mathbb{A} \quad \frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_2 \mid \alpha_1 : d \wedge \alpha_2 : d_2 \wedge \text{broadcastable}(d_2, d)} \text{CONV-ARRBINOP}_R}{\frac{\exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1}{\exists \alpha'_1 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha'_1) \oplus (\alpha_2)} \text{E-PROPARRBINOP}_L} \text{ARRAY-PROGRESS} \\
\frac{\exists d_2 \mid \alpha_2 : d_2 \quad \exists d_2 \mid \text{broadcastable}(d_2, d) \quad \frac{\alpha_1 \subset (\alpha_1) \oplus (\alpha_2) \quad \exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1 \quad \alpha_1 : d}{\exists \alpha'_1 \mid \alpha'_1 : d} \text{ARRAY-PRESHYP}}{\frac{\exists \alpha'_1 \mid (\alpha'_1) \oplus (\alpha_2) : d}{\exists \alpha'_1 \mid (\alpha'_1) \oplus (\alpha_2) : d} \text{ARRAYBINOP}_L} \text{ARRAYBINOP}_L
\end{array}$$

We have shown that $(\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha'_1) \oplus (\alpha_2)$ and $(\alpha'_1) \oplus (\alpha_2) : d$. $(\alpha_1) \oplus (\alpha_2)$ also has the type d . Therefore, preservation is satisfied for this case.

Case 3: $\alpha = (\alpha_1) \oplus (\alpha_2)$ and $\alpha_2 \notin \mathbb{A}$. Similar to case 2.

$$\begin{array}{c}
\frac{\alpha_2 \notin \mathbb{A} \quad \frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_1 \mid \alpha_1 : d_1 \wedge \alpha_2 : d \wedge \text{broadcastable}(d_1, d)} \text{CONV-ARRBINOP}_L}{\frac{\exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2}{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha_1) \oplus (\alpha'_2)} \text{E-PROPARRBINOP}_R} \text{ARRAY-PROGRESS} \\
\frac{\exists d_1 \mid \alpha_1 : d_1 \quad \exists d_1 \mid \text{broadcastable}(d_1, d) \quad \frac{\alpha_2 \subset (\alpha_1) \oplus (\alpha_2) \quad \exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2 \quad \alpha_2 : d}{\exists \alpha'_2 \mid \alpha'_2 : d} \text{ARRAY-PRESHYP}}{\frac{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha'_2) : d}{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha'_2) : d} \text{ARRAYBINOP}_R} \text{ARRAYBINOP}_R
\end{array}$$

Since CONV-ARRBINOP_L is used in the above proof, we must also show that the conclusion holds when CONV-ARRBINOP_R is used instead.

$$\begin{array}{c}
\frac{\alpha_2 \notin \mathbb{A} \quad \frac{(\alpha_1) \oplus (\alpha_2) : d}{\exists d_2 \mid \alpha_1 : d \wedge \alpha_2 : d_2 \wedge \text{broadcastable}(d_2, d)} \text{CONV-ARRBINOP}_R}{\frac{\exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2}{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha_1) \oplus (\alpha'_2)} \text{E-PROPARRBINOP}_L} \text{ARRAY-PROGRESS} \\
\frac{\alpha_1 : d \quad \exists d_2 \mid \text{broadcastable}(d_2, d) \quad \frac{\alpha_2 \subset (\alpha_1) \oplus (\alpha_2) \quad \exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2 \quad \alpha_2 : d}{\exists \alpha'_2, d_2 \mid \alpha'_2 : d_2} \text{ARRAY-PRESHYP}}{\frac{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha'_2) : d}{\exists \alpha'_2 \mid (\alpha_1) \oplus (\alpha'_2) : d} \text{ARRAYBINOP}_L} \text{ARRAYBINOP}_L
\end{array}$$

We have shown that $(\alpha_1) \oplus (\alpha_2) \rightarrow (\alpha_1) \oplus (\alpha'_2)$ and $(\alpha_1) \oplus (\alpha'_2) : d$. $(\alpha_1) \oplus (\alpha_2)$ also has the type d . Therefore, preservation is satisfied for this case.

Case 4: $\alpha = (\alpha_1) \oplus (\alpha_2)$ and $\alpha_1, \alpha_2 \in \mathbb{A}$. We know from case 4 of the progress proof that $(\alpha_1) \oplus (\alpha_2) \rightarrow a_3$ and that $a_3 : d$. Since $(\alpha_1) \oplus (\alpha_2)$ also has the dimension d , the preservation theorem is satisfied for this case.

Case 5: $\alpha = \text{matmul}(\alpha_1, \alpha_2)$ and $\alpha_1 \notin \mathbb{A}$.

$$\begin{array}{c}
\frac{\alpha_1 \notin \mathbb{A} \quad \frac{\text{matmul}(\alpha_1, \alpha_2) : d}{\exists d_1, d_2, d_i \mid d = d_1 \times d_2 \wedge \alpha_1 : d_1 \times d_i \wedge \alpha_2 : d_i \times d_2} \text{CONV-ARRAYMATMUL}}{\frac{\exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1}{\exists \alpha'_1 \mid \text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha'_1, \alpha_2)} \text{E-PROPARRMATMULL}} \text{ARRAYPROGRESS} \\
\frac{\exists d_1, d_2 \mid d = d_1 \times d_2 \quad \frac{\alpha_1 \in \text{matmul}(\alpha_1, \alpha_2) \quad \exists \alpha'_1 \mid \alpha_1 \rightarrow \alpha'_1 \quad \exists d_1, d_i \mid \alpha_1 : d_1 \times d_i}{\exists \alpha'_1, d_1, d_i \mid \alpha'_1 : d_1 \times d_i} \text{ARRAY-PRESHYP}}{\frac{\exists \alpha'_1, d_1, d_2 \mid \text{matmul}(\alpha'_1, \alpha_2) : d_1 \times d_2}{\exists \alpha'_1 \mid \text{matmul}(\alpha'_1, \alpha_2) : d} \text{SUBST}} \text{ARRAYMATMUL}
\end{array}$$

Since $\text{matmul}(\alpha_1, \alpha_2) : d$, $\text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha'_1, \alpha_2)$, and $\text{matmul}(\alpha'_1, \alpha_2) : d$, preservation holds for this case.

Case 6: $\alpha = \text{matmul}(\alpha_1, \alpha_2)$ and $\alpha_2 \notin \mathbb{A}$.

$$\begin{array}{c}
\frac{\alpha_2 \notin \mathbb{A} \quad \frac{\text{matmul}(\alpha_1, \alpha_2) : d}{\exists d_1, d_2, d_i \mid d = d_1 \times d_2 \wedge \alpha_1 : d_1 \times d_i \wedge \alpha_2 : d_i \times d_2} \text{CONV-ARRAYMATMUL}}{\frac{\exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2}{\exists \alpha'_2 \mid \text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha_1, \alpha'_2)} \text{E-PROPARRMATMULL}} \text{ARRAY-PROGRESS} \\
\\
\frac{\frac{\frac{\alpha_2 \in \text{matmul}(\alpha_1, \alpha_2) \quad \exists \alpha'_2 \mid \alpha_2 \rightarrow \alpha'_2 \quad \exists d_2, d_i \mid \alpha_2 : d_i \times d_2}{\exists \alpha'_2, d_2, d_i \mid \alpha'_2 : d_i \times d_2} \text{ARRAY-PRESHYP}}{\frac{\exists d_1, d_i \mid \alpha_1 : d_1 \times d_i \quad \exists \alpha'_2, d_2, d_i \mid \alpha'_2 : d_i \times d_2}{\exists \alpha'_2, d_1, d_2 \mid \text{matmul}(\alpha_1, \alpha'_2) : d_1 \times d_2} \text{ARRAYMATMUL}}}{\frac{\exists d_1, d_2 \mid d = d_1 \times d_2 \quad \exists \alpha'_2, d_1, d_2 \mid \text{matmul}(\alpha_1, \alpha'_2) : d_1 \times d_2}{\exists \alpha'_2 \mid \text{matmul}(\alpha_1, \alpha'_2) : d} \text{SUBST}}
\end{array}$$

Since $\text{matmul}(\alpha_1, \alpha_2) : d$, $\text{matmul}(\alpha_1, \alpha_2) \rightarrow \text{matmul}(\alpha_1, \alpha'_2)$, and $\text{matmul}(\alpha_1, \alpha'_2) : d$, preservation holds for this case.

Case 7: $\alpha = \text{matmul}(\alpha_1, \alpha_2)$ and $\alpha_1, \alpha_2 \in \mathbb{A}$. In case 7 of the progress proof, we showed that $\text{matmul}(\alpha_1, \alpha_2) \rightarrow \alpha_3$, and that $\alpha_3 : d_1 \times d_2$. We also showed that $d = d_1 \times d_2$. Therefore, $\alpha_3 : d$. Since $\text{matmul}(\alpha_1, \alpha_2) : d$, $\text{matmul}(\alpha_1, \alpha_2) \rightarrow \alpha_3$, and $\alpha_3 : d$, preservation holds for this case.

Summary. The seven cases that we prove preservation for are comprehensive. In each case, we show that, if the corresponding array expression can be evaluated further, the type of the array expression remains unchanged after evaluation. The preservation theorem holds for array expressions.

4.3 Progress

Theorem. A typed value v is either a basic value of the form $\tau(a)$ or can take an evaluation step to some v' . The theorem is more formally stated below:

$$\frac{v \notin v_B \quad v : (\tau, d)}{\exists v' \mid v \rightarrow v'} \text{PROGRESS}$$

We will utilize structural induction to prove the progress theorem. Our inductive hypothesis is as follows:

$$\frac{v_i \subset v \quad v_i \notin v_B \quad v : (\tau, d)}{\exists v'_i \mid v_i \rightarrow v'_i} \text{PROG-INDHYP}$$

We will show that the theorem of progress holds for each SMALLERSPACE rule that can type check a value.

4.3.1 Init Progress

$$\frac{\alpha : d \times \text{channel_count}(\tau)}{\tau(\alpha) : (\tau, d)} \text{INIT}$$

Case 1: $\alpha \in \mathbb{A}$. If α is an array literal, $\tau(\alpha)$ is a basic value and cannot be evaluated further. Therefore, the progress theorem holds for this case.

Case 2: $\alpha \notin \mathbb{A}$. We have proven ARRAY-PROGRESS. As such, we know that if $\alpha \notin \mathbb{A}$, there must exist some α' such that $\alpha \rightarrow \alpha'$. Through E-PROPINIT, we can conclude that $\tau(\alpha) \rightarrow \tau(\alpha')$. Therefore, VALUE-PROGRESS holds for case 2.

Summary. Since the progress theorem holds for all cases of the INIT rule, the INIT rule satisfies the progress theorem. Any non-basic value that is type checked by the INIT rule can be further evaluated.

4.3.2 Cast Progress

$$\frac{v : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\tau_2(v) : (\tau_2, d)} \text{CAST}$$

Case 1: $v \notin v_B$. If $v \notin v_B$, we can conclude by the inductive hypothesis and E-PROPCAST that there exists some v' such that $\tau_2(v) \rightarrow \tau_2(v')$.

Case 2: $v \in v_B, \tau_1 = \tau_2$.

$$\frac{\frac{v \in v_B \quad \frac{\tau_2(v) : (\tau_2, d)}{\exists \tau_1 \mid v : (\tau_1, d)} \text{CONV-CAST}}{\exists \tau_1, a_1 : d \times \text{channel_count}(\tau_1) \mid v = \tau_1(a_1)} \text{CONV-BASICINIT} \quad \frac{\tau_1 = \tau_2 \quad \frac{\tau_1(\tau_1(a_1)) \rightarrow \tau_1(a_1)}{\tau_2(\tau_1(a_1)) \rightarrow \tau_2(a_1)} \text{E-CASTTRIVIAL}}{\tau_2(\tau_1(a_1)) \rightarrow \tau_2(a_1)} \text{SUBST}}{\exists \tau_1, a_1 : d \times \text{channel_count}(\tau_1) \mid \tau_2(v) \rightarrow \tau_2(a_1)} \text{SUBST}$$

Since $\tau_2(v) \rightarrow \tau_2(a_1)$, progress is fulfilled for this case.

Case 3: $v \in v_B, \tau_1$ adjacent to τ_2 . If τ_1 and τ_2 are adjacent in the casting graph and $\text{path_exists}(\tau_1, \tau_2)$, there exists a corresponding adjacent casting rule. A list of these rules can be found in Fig. 7. We can apply these rules to show that there exists some α such that $\tau_2(v) \rightarrow \tau_2(\alpha)$. This shows that VALUE-PROGRESS holds for this case. We will show one example of this case, where $\tau_1 = \tau_{\text{Light}}$ and $\tau_2 = \tau_{\text{LMS}}$:

$$\frac{\frac{v \in v_B \quad \frac{\tau_1 = \tau_{\text{Light}} \quad \frac{\tau_2(v) : (\tau_2, d)}{\exists \tau_1 \mid v : (\tau_1, d)} \text{CONV-CAST}}{v : (\tau_{\text{Light}}, d)} \text{SUBST}}{\exists a : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v = \tau_{\text{Light}}(a)} \text{CONV-BASICINIT} \quad \frac{\tau_2 = \tau_{\text{LMS}} \quad \frac{\tau_{\text{LMS}}(\tau_{\text{Light}}(a)) \rightarrow \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{LIGHT2LMS}}))}{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_2(\text{matmul}(a, M_{\text{LIGHT2LMS}}))} \text{E-LIGHT2LMS}}{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_2(\text{matmul}(a, M_{\text{LIGHT2LMS}}))} \text{SUBST}}{\tau_2(v) \rightarrow \tau_2(\text{matmul}(a, M_{\text{LIGHT2LMS}}))} \text{SUBST}$$

Since $\tau_2(v) \rightarrow \tau_2(\text{matmul}(a, M_{\text{LIGHT2LMS}}))$, the progress theorem holds for this sub-case. Other instances of this case (i.e. $\tau_1 = \tau_{\text{XYZ}}, \tau_2 = \tau_{\text{sRGB}}$) come to the same conclusion.

Case 4: $v \in v_B$, τ_1 **not adjacent to** τ_2 . From CONV-CAST, we know that $\text{path_exists}(\tau_1, \tau_2)$. We also know that in this case there is at least 1 hop that needs to be made between τ_1 and τ_2 on the casting graph. Since the casting graph is a directed forest, there is a unique shortest path between τ_1 and τ_2 . Therefore, E-CASTHOP will be able to find some τ_i such that $\tau_2(v) \rightarrow \tau_2(\tau_i(v))$. We will show one example where $\tau_1 = \tau_{\text{Light}}$ and $\tau_2 = \tau_{\text{XYZ}}$:

$$\begin{array}{c}
\frac{\text{path_exists}(\tau_{\text{Light}}, \tau_{\text{XYZ}}) \quad 1 \leq \text{distance}(\tau_{\text{LMS}}, \tau_{\text{XYZ}}) \leq \text{distance}(\tau_{\text{Light}}, \tau_{\text{XYZ}}) \quad \text{distance}(\tau_{\text{Light}}, \tau_{\text{LMS}}) = 1}{\tau_2 = \tau_{\text{XYZ}} \quad \frac{\tau_{\text{XYZ}}(\tau_{\text{Light}}(a)) \rightarrow \tau_{\text{XYZ}}(\tau_{\text{LMS}}(\tau_{\text{Light}}(a)))}{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_2(\tau_{\text{LMS}}(\tau_{\text{Light}}(a)))} \text{SUBST}} \text{E-CASTHOP} \\
\\
\frac{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_2(\tau_{\text{LMS}}(\tau_{\text{Light}}(a))) \quad \frac{\tau_2(v) : (\tau_2, d)}{\exists \tau_1 \mid v : (\tau_1, d)} \text{CONV-CAST}}{\frac{v \in v_B \quad v : (\tau_{\text{Light}}, d)}{\exists a \times \text{channel_count}(\tau_{\text{Light}}) \mid v = \tau_{\text{Light}}(a)} \text{SUBST}} \text{CONV-BASICINIT} \\
\frac{\tau_2(v) \rightarrow \tau_2(\tau_{\text{LMS}}(v))}{\tau_2(v) \rightarrow \tau_2(\tau_{\text{LMS}}(v))} \text{SUBST}
\end{array}$$

Since $\tau_2(v) \rightarrow \tau_2(\tau_{\text{LMS}}(v))$, the progress theorem holds for this sub-case. Other instances of this case (i.e. $\tau_1 = \tau_{\text{LMS}}$, $\tau_2 = \tau_{\text{sRGB}}$) come to the same conclusion.

Summary. Since the progress theorem holds for all cases of the CAST rule, the CAST rule satisfies the progress theorem. Any non-basic value that is type checked by the CAST rule can be further evaluated.

4.3.3 TristimulusAdd Progress

$$\frac{v_1, v_2 : (\tau_{\text{tristimulus}}, d)}{v_1 + v_2 : (\tau_{\text{tristimulus}}, d)} \text{TRISTIMULUSADD}$$

Case 1: $v_1 \notin v_B$ **and/or** $v_2 \notin v_B$. By the inductive hypothesis, there must exist some v'_1 such that $v_1 \rightarrow v'_1$ or some v'_2 such that $v_2 \rightarrow v'_2$. By E-PROPBINOPL or E-PROPBINOPLR, we can conclude that $v_1 + v_2 \rightarrow v'_1 + v_2$ or $v_1 + v_2 \rightarrow v_1 + v'_2$. Therefore, progress holds for this case.

Case 2: $v_1, v_2 \in v_B$. We can see from the CONV-TRISTIMULUSADD rule that both v_1 and v_2 must share the same shaped type. A different evaluation rule may apply to the value depending on what tristimulus type v_1 and v_2 share. We will exhaustively cover every possible tristimulus type case.

Case 2-a: $v_1, v_2 : (\tau_{\text{XYZ}}, d)$

$$\begin{array}{c}
\frac{v_1 \in v_B \quad \frac{v_1 + v_2 : (\tau_{\text{XYZ}}, d)}{v_1 : (\tau_{\text{XYZ}}, d)} \text{CONV-TRISTIMULUSADD}}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 = \tau_{\text{XYZ}}(a_1)} \text{CONV-BASICINIT} \\
\\
\frac{v_2 \in v_B \quad \frac{v_1 + v_2 : (\tau_{\text{XYZ}}, d)}{v_2 : (\tau_{\text{XYZ}}, d)} \text{CONV-TRISTIMULUSADD}}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_2 = \tau_{\text{XYZ}}(a_2)} \text{CONV-BASICINIT} \\
\\
\frac{\frac{\tau_{\text{XYZ}}(a_1) + \tau_{\text{XYZ}}(a_2) \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 + \tau_{\text{XYZ}}(a_2) \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))} \text{E-XYZADD} \quad \exists a_1 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 = \tau_{\text{XYZ}}(a_1)}{\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 + v_2 \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))} \text{SUBST} \\
\\
\frac{\exists a_1 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 + \tau_{\text{XYZ}}(a_2) \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2)) \quad \exists a_2 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_2 = \tau_{\text{XYZ}}(a_2)}{\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 + v_2 \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))} \text{SUBST}
\end{array}$$

Since $v_1 + v_2 \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))$, progress holds for this case.

Case 2-b: $v_1, v_2 : (\tau_{\text{LMS}}, d)$

$$\begin{array}{c}
\frac{v_1 + v_2 : (\tau_{\text{LMS}}, d)}{v_1 \in v_B} \text{ CONV-TRISTIMULUSADD} \\
\frac{v_1 : (\tau_{\text{LMS}}, d)}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 = \tau_{\text{LMS}}(a_1)} \text{ CONV-BASICINIT} \\
\\
\frac{v_1 + v_2 : (\tau_{\text{LMS}}, d)}{v_2 \in v_B} \text{ CONV-TRISTIMULUSADD} \\
\frac{v_2 : (\tau_{\text{LMS}}, d)}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_2 = \tau_{\text{LMS}}(a_2)} \text{ CONV-BASICINIT} \\
\\
\frac{\tau_{\text{LMS}}(a_1) + \tau_{\text{LMS}}(a_2) \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 + \tau_{\text{LMS}}(a_2) \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))} \text{ E-LMSADD} \\
\frac{\exists a_1 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 = \tau_{\text{LMS}}(a_1)}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 + \tau_{\text{LMS}}(a_2) \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))} \text{ SUBST} \\
\\
\frac{\exists a_1 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 + \tau_{\text{LMS}}(a_2) \rightarrow \tau_{\text{LMS}}((a_1) + (a_2)) \quad \exists a_2 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_2 = \tau_{\text{LMS}}(a_2)}{\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 + v_2 \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))} \text{ SUBST}
\end{array}$$

Since $v_1 + v_2 \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))$, progress holds for this case.

Case 2-c: $v_1, v_2 : (\tau_{\text{sRGB}}, d)$

$$\begin{array}{c}
\frac{v_1 + v_2 : (\tau_{\text{sRGB}}, d)}{v_1 \in v_B} \text{ CONV-TRISTIMULUSADD} \\
\frac{v_1 : (\tau_{\text{sRGB}}, d)}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{sRGB}}) \mid v_1 = \tau_{\text{sRGB}}(a_1)} \text{ CONV-BASICINIT} \\
\\
\frac{v_1 + v_2 : (\tau_{\text{sRGB}}, d)}{v_2 \in v_B} \text{ CONV-TRISTIMULUSADD} \\
\frac{v_2 : (\tau_{\text{sRGB}}, d)}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{sRGB}}) \mid v_2 = \tau_{\text{sRGB}}(a_2)} \text{ CONV-BASICINIT} \\
\\
\frac{\tau_{\text{sRGB}}(a_1) + \tau_{\text{sRGB}}(a_2) \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * ((\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])))}{\tau_{\text{sRGB}}(a_1) + \tau_{\text{sRGB}}(a_2) \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * ((\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])))} \text{ E-sRGBADD} \\
\\
\frac{v_1 : (\tau_{\text{sRGB}}, d) \quad \tau_{\text{sRGB}}(a_1) + \tau_{\text{sRGB}}(a_2) \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * ((\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])))}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{sRGB}}) \mid v_1 + \tau_{\text{sRGB}}(a_2) \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * ((\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])))} \text{ SUBST} \\
\\
\frac{v_2 : (\tau_{\text{sRGB}}, d) \quad v_1 + \tau_{\text{sRGB}}(a_2) \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * ((\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])))}{\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{sRGB}}) \mid v_1 + v_2 \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * ((\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])))} \text{ SUBST}
\end{array}$$

Since $v_1 + v_2 \rightarrow \tau_{\text{sRGB}}(\dots)$, progress holds for this case.

Summary. Since the progress theorem holds for all cases of the TRISTIMULUSADD rule, the TRISTIMULUSADD rule satisfies the progress theorem. Any non-basic value that is type checked by the TRISTIMULUSADD rule can be further evaluated.

4.3.4 LightAdd Progress

$$\frac{v_1, v_2 : (\tau_{\text{Light}}, d)}{v_1 + v_2 : (\tau_{\text{Light}}, d)} \text{ LIGHTADD}$$

Case 1: $v_1 \notin v_B$ and/or $v_2 \notin v_B$. By the inductive hypothesis, there must exist some v'_1 such that $v_1 \rightarrow v'_1$ or some v'_2 such that $v_2 \rightarrow v'_2$. By E-PROPBINOPL or E-PROPBINOPR, we can conclude that $v_1 + v_2 \rightarrow v'_1 + v_2$ or $v_1 + v_2 \rightarrow v_1 + v'_2$. Therefore, progress holds for this case.

Case 2: $v_1, v_2 \in v_B$.

$$\begin{array}{c}
\frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_1 \in v_B \quad \frac{}{v_1 : (\tau_{\text{Light}}, d)} \text{ CONV-LIGHTADD}} \text{ CONV-BASICINIT} \\
\frac{}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 = \tau_{\text{Light}}(a_1)} \\
\\
\frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_2 \in v_B \quad \frac{}{v_2 : (\tau_{\text{Light}}, d)} \text{ CONV-LIGHTADD}} \text{ CONV-BASICINIT} \\
\frac{}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_2 = \tau_{\text{Light}}(a_2)} \\
\\
\frac{\tau_{\text{Light}}(a_1) + \tau_{\text{Light}}(a_2) \rightarrow \tau_{\text{Light}}((a_1) + (a_2)) \quad \frac{}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 = \tau_{\text{Light}}(a_1)} \text{ E-LIGHTADD}}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 + \tau_{\text{Light}}(a_2) \rightarrow \tau_{\text{Light}}((a_1) + (a_2))} \text{ SUBST} \\
\\
\frac{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 + \tau_{\text{Light}}(a_2) \rightarrow \tau_{\text{Light}}((a_1) + (a_2)) \quad \frac{}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_2 = \tau_{\text{Light}}(a_2)} \text{ SUBST}}{\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 + v_2 \rightarrow \tau_{\text{Light}}((a_1) + (a_2))} \text{ SUBST}
\end{array}$$

Since $v_1 + v_2 \rightarrow \tau_{\text{Light}}((a_1) + (a_2))$, progress holds for this case.

Summary. Since the progress theorem holds for both cases of the LIGHTADD rule, the LIGHTADD rule satisfies the progress theorem. Any non-basic value that is type checked by the LIGHTADD rule can be further evaluated.

4.3.5 Reflect Progress

$$\frac{v_1 : (\tau_{\text{Light}}, d) \quad v_2 : (\tau_{\text{Reflectance}}, d)}{v_1 \times v_2 : (\tau_{\text{Light}}, d)} \text{ REFLECT}$$

Case 1: $v_1 \notin v_B$ and/or $v_2 \notin v_B$. By the inductive hypothesis, there must exist some v'_1 such that $v_1 \rightarrow v'_1$ or some v'_2 such that $v_2 \rightarrow v'_2$. By E-PROPBINOP or E-PROPBINOPR, we can conclude that $v_1 \times v_2 \rightarrow v'_1 \times v_2$ or $v_1 \times v_2 \rightarrow v_1 \times v'_2$. Therefore, progress holds for this case.

Case 2: $v_1, v_2 \notin v_B$.

$$\begin{array}{c}
\frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_1 \in v_B \quad \frac{}{v_1 : (\tau_{\text{Light}}, d)} \text{ CONV-REFLECT}} \text{ CONV-BASICINIT} \\
\frac{}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 = \tau_{\text{Light}}(a_1)} \\
\\
\frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_2 \in v_B \quad \frac{}{v_2 : (\tau_{\text{Reflectance}}, d)} \text{ CONV-REFLECT}} \text{ CONV-BASICINIT} \\
\frac{}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{Reflectance}}) \mid v_2 = \tau_{\text{Reflectance}}(a_2)} \\
\\
\frac{\tau_{\text{Light}}(a_1) \times \tau_{\text{Reflectance}}(a_2) \rightarrow \tau_{\text{Light}}((a_1) \times (a_2)) \quad \frac{}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 = \tau_{\text{Light}}(a_1)} \text{ E-REFLECT}}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 \times \tau_{\text{Reflectance}}(a_2) \rightarrow \tau_{\text{Light}}((a_1) \times (a_2))} \text{ SUBST} \\
\\
\frac{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 \times \tau_{\text{Reflectance}}(a_2) \rightarrow \tau_{\text{Light}}((a_1) \times (a_2)) \quad \frac{}{\exists a_2 : d \times \text{channel_count}(\tau_{\text{Reflectance}}) \mid v_2 = \tau_{\text{Reflectance}}(a_2)} \text{ SUBST}}{\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}), a_2 : d \times \text{channel_count}(\tau_{\text{Reflectance}}) \mid v_1 \times v_2 \rightarrow \tau_{\text{Light}}((a_1) \times (a_2))} \text{ SUBST}
\end{array}$$

Since $v_1 \times v_2 \rightarrow \tau_{\text{Light}}((a_1) \times (a_2))$, progress holds for this case.

Summary. Since the progress theorem holds for both cases of the LIGHTADD rule, the LIGHTADD rule satisfies the progress theorem. Any non-basic value that is type checked by the LIGHTADD rule can be further evaluated.

4.3.6 Progress Summary

We have shown that each SMALLERSPACE type rule fulfills the progress theorem by the principle of induction. Therefore, any type checked SMALLERSPACE non-basic value can be evaluated further. SMALLERSPACE satisfies the progress theorem.

4.4 Preservation Theorem

Theorem. A non-basic typed value v can be evaluated to v' according to the progress theorem. $v : (\tau, d)$ implies $v' : (\tau, d)$. The theorem is more formally stated below:

$$\frac{v : (\tau, d) \quad v \rightarrow v'}{v' : (\tau, d)} \text{PRESERVATION}$$

We will utilize structural induction to prove the preservation theorem. The inductive hypothesis is as follows:

$$\frac{v_i \subset v \quad v_i : (\tau_i, d_i) \quad v_i \rightarrow v'_i}{v'_i : (\tau_i, d_i)} \text{PRES-INDHYP}$$

4.4.1 Init Preservation

$$\frac{\alpha : d \times \text{channel_count}(\tau)}{\tau(\alpha) : (\tau, d)} \text{INIT}$$

Case 1: $\alpha \in \mathbb{A}$. If α is an array literal, $\tau(\alpha)$ is a basic value and cannot be evaluated further. Therefore, the preservation theorem holds for this case.

Case 2: $\alpha \notin \mathbb{A}$.

$$\begin{array}{c} \frac{\alpha \notin \mathbb{A}}{\exists \alpha' \mid \alpha \rightarrow \alpha'} \text{ARRAY-PROGRESS} \quad \frac{\tau(\alpha) : (\tau, d)}{\alpha : d \times \text{channel_count}(\tau)} \text{CONV-INIT} \\ \hline \frac{\exists \alpha' : d \times \text{channel_count}(\tau) \mid \alpha \rightarrow \alpha'}{\exists \alpha' : d \times \text{channel_count}(\tau) \mid \tau(\alpha) \rightarrow \tau(\alpha')} \text{ARRAY-PRESERVATION} \\ \hline \frac{\alpha' : d \times \text{channel_count}(\tau)}{\tau(\alpha') : (\tau, d)} \text{E-PROPINIT} \end{array}$$

Since $\tau(\alpha) : (\tau, d)$, $\tau(\alpha) \rightarrow \tau(\alpha')$, and $\tau(\alpha') : (\tau, d)$, preservation holds for this case.

Summary. Preservation holds for both cases of the INIT rule. Any non-basic type that INIT type checks can be evaluated further (PROGRESS). The type of the non-basic value is preserved after the evaluation step.

4.4.2 Cast Preservation

$$\frac{v : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\tau_2(v) : (\tau_2, d)} \text{CAST}$$

Case 1: $v \notin v_B$. If $v \notin v_B$, we know that there exists some v' such that $\tau_2(v) \rightarrow \tau_2(v')$ by the progress theorem. By the inductive hypothesis, v has the same type as v' .

$$\frac{v : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\tau_2(v) : (\tau_2, d)} \text{CAST}$$

$$\frac{v' : (\tau_1, d) \quad \text{path_exists}(\tau_1, \tau_2)}{\tau_2(v') : (\tau_2, d)} \text{CAST}$$

Since $\tau_2(v) : (\tau_2, d)$, $\tau_2(v) \rightarrow \tau_2(v')$, and $\tau_2(v') : (\tau_2, d)$, preservation holds for this case.

case 2: $v \in v_B$, $\tau_1 = \tau_2$.

$$\begin{array}{c}
\frac{\tau_2(v) : (\tau_2, d)}{v \in v_B \quad \exists \tau_1 \mid v : (\tau_1, d)} \text{ CONV-CAST} \\
\frac{\exists \tau_1, a_1 : d \times \text{channel_count}(\tau_1) \mid v = \tau_1(a_1)}{\exists \tau_1, a_1 : d \times \text{channel_count}(\tau_1) \mid \tau_2(v) \rightarrow \tau_2(a_1)} \text{ CONV-BASICINIT} \quad \frac{\tau_1 = \tau_2 \quad \tau_1(\tau_1(a_1)) \rightarrow \tau_1(a_1)}{\tau_2(\tau_1(a_1)) \rightarrow \tau_2(a_1)} \text{ E-CASTTRIVIAL} \\
\text{SUBST}
\end{array}$$

$$\frac{\frac{a_1 : d \times \text{channel_count}(\tau_1)}{a_1 : d \times \text{channel_count}(\tau_2)} \text{ SUBST} \quad \tau_1 = \tau_2}{\tau_2(a_1) : (\tau_2, d)} \text{ INIT}$$

Since $\tau_2(v) : (\tau_2, d)$, $\tau_2(v) \rightarrow \tau_2(a_1)$, and $\tau_2(a_1) : (\tau_2, d)$, preservation holds for this case.

Case 3: $v \in v_B$, τ_1 **adjacent to** τ_2 . We know that there exists a corresponding adjacent casting rule for τ_1 and τ_2 . We will show that preservation holds for $\tau_1 = \tau_{\text{Light}}$ and $\tau_2 = \tau_{\text{LMS}}$. Other proofs for this case are similar in form.

$$\begin{array}{c}
\frac{\tau_2(v) : (\tau_2, d)}{\tau_1 = \tau_{\text{Light}} \quad \exists \tau_1 \mid v : (\tau_1, d)} \text{ CONV-CAST} \\
\frac{v \in v_B \quad v : (\tau_{\text{Light}}, d)}{\exists a : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v = \tau_{\text{Light}}(a)} \text{ SUBST} \quad \frac{\tau_2 = \tau_{\text{LMS}} \quad \tau_{\text{LMS}}(\tau_{\text{Light}}(a)) \rightarrow \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}}))}{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}}))} \text{ E-LIGHT2LMS} \\
\text{SUBST}
\end{array}$$

$$\begin{array}{c}
\frac{\text{channel_count}(\tau_{\text{LMS}}) = 3 \quad \frac{M_{\text{Light2LMS}} : 89 \times 3 \quad \frac{a : d \times \text{channel_count}(\tau_{\text{Light}}) \quad \text{channel_count}(\tau_{\text{Light}}) = 89}{a : d \times 89} \text{ SUBST}}{\text{matmul}(a, M_{\text{Light2LMS}}) : d \times 3} \text{ E-ARRMATMUL} \\
\frac{\text{matmul}(a, M_{\text{Light2LMS}}) : d \times \text{channel_count}(\tau_{\text{LMS}})}{\tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}})) : (\tau_{\text{LMS}}, d)} \text{ INIT} \\
\frac{\tau_2 = \tau_{\text{LMS}} \quad \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}})) : (\tau_{\text{LMS}}, d)}{\tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}})) : (\tau_2, d)} \text{ SUBST}
\end{array}$$

Since $\tau_2(v) : (\tau_2, d)$, $\tau_2(v) \rightarrow \tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}}))$, and $\tau_{\text{LMS}}(\text{matmul}(a, M_{\text{Light2LMS}})) : (\tau_2, d)$, preservation holds for this sub-case. Other sub-cases (i.e., $\tau_1 = \tau_{\text{XYZ}}$ and $\tau_2 = \tau_{\text{LMS}}$) come to the same conclusion.

Case 4: $v \in v_B$, τ_1 **not adjacent to** τ_2 . Like case 4 of the CAST progress proof, we will demonstrate this case for when $\tau_1 = \tau_{\text{Light}}$ and $\tau_2 = \tau_{\text{XYZ}}$. The proof for other pairs of types are similar in form.

$$\begin{array}{c}
\frac{\text{path_exists}(\tau_{\text{Light}}, \tau_{\text{XYZ}}) \quad 1 \leq \text{distance}(\tau_{\text{LMS}}, \tau_{\text{XYZ}}) \leq \text{distance}(\tau_{\text{Light}}, \tau_{\text{XYZ}}) \quad \text{distance}(\tau_{\text{Light}}, \tau_{\text{LMS}}) = 1}{\tau_2 = \tau_{\text{XYZ}} \quad \frac{\tau_{\text{XYZ}}(\tau_{\text{Light}}(a)) \rightarrow \tau_{\text{XYZ}}(\tau_{\text{LMS}}(\tau_{\text{Light}}(a)))}{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_2(\tau_{\text{LMS}}(\tau_{\text{Light}}(a)))} \text{ SUBST}} \text{ E-CASTHOP}
\end{array}$$

$$\begin{array}{c}
\frac{\tau_2(v) : (\tau_2, d)}{\tau_1 = \tau_{\text{Light}} \quad \exists \tau_1 \mid v : (\tau_1, d)} \text{ CONV-CAST} \\
\frac{v \in v_B \quad v : (\tau_{\text{Light}}, d)}{\exists a \times \text{channel_count}(\tau_{\text{Light}}) \mid v = \tau_{\text{Light}}(a)} \text{ SUBST} \quad \frac{\tau_2(\tau_{\text{Light}}(a)) \rightarrow \tau_2(\tau_{\text{LMS}}(\tau_{\text{Light}}(a)))}{\tau_2(v) \rightarrow \tau_2(\tau_{\text{LMS}}(v))} \text{ CONV-BASICINIT}
\end{array}$$

$$\begin{array}{c}
\frac{\tau_1 = \tau_{\text{Light}} \quad \text{path_exists}(\tau_{\text{Light}}, \tau_{\text{LMS}})}{v : (\tau_1, d) \quad \frac{\text{path_exists}(\tau_1, \tau_{\text{LMS}})}{\tau_{\text{LMS}}(v) : (\tau_{\text{LMS}}, d)} \text{ CAST}} \text{ SUBST} \quad \frac{\tau_2 = \tau_{\text{XYZ}} \quad \text{path_exists}(\tau_{\text{LMS}}, \tau_{\text{XYZ}})}{\text{path_exists}(\tau_{\text{LMS}}, \tau_2)} \text{ SUBST} \\
\text{CAST}
\end{array}$$

Since $\tau_2(v) : (\tau_2, d)$, $\tau_2(v) \rightarrow \tau_2(\tau_{\text{LMS}}(v))$, and $\tau_2(\tau_{\text{LMS}}(v)) : (\tau_2, d)$, preservation holds for this sub-case. Other sub-cases (i.e., $\tau_1 = \tau_{\text{LMS}}$ and $\tau_2 = \tau_{\text{SRGB}}$) come to the same conclusion.

Summary. Preservation holds for all cases of the CAST rule. Any non-basic type that CAST type checks can be evaluated further (PROGRESS). The type of the non-basic value is preserved after the evaluation step.

4.4.3 TristimulusAdd Preservation

$$\frac{v_1, v_2 : (\tau_{\text{tristimulus}}, d)}{v_1 + v_2 : (\tau_{\text{tristimulus}}, d)} \text{TRISTIMULUSADD}$$

Case 1: $v_1 \notin v_B$ and/or $v_2 \notin v_B$. We know from case 1 of the TRISTIMULUSADD progress proof that there exists some v'_1 or v'_2 such that $v_1 + v_2 \rightarrow v'_1 + v_2$ or $v_1 + v_2 \rightarrow v_1 + v'_2$. By the inductive hypothesis we can conclude that v'_1 and v'_2 have the same shaped types as v_1 and v_2 . Therefore, by the TRISTIMULUSADD rule, $v_1 + v_2$ is the same type as $v'_1 + v_2$ or $v_1 + v'_2$. Thus, preservation holds for case 1.

Case 2: $v_1, v_2 \in v_B$. We can see from the CONV-TRISTIMULUSADD rule that both v_1 and v_2 must share the same shaped type. A different evaluation rule may apply to the value depending on what tristimulus type v_1 and v_2 share. We will exhaustively cover every possible tristimulus type case.

Case 2-a: $v_1, v_2 : (\tau_{\text{XYZ}}, d)$ We know from case 2-a of the progress proof for TRISTIMULUSADD that $\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \mid v_1 + v_2 \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))$.

$$\frac{\frac{a_1, a_2 : d \times \text{channel_count}(\tau_{\text{XYZ}}) \quad \frac{\text{broadcastable}(d \times \text{channel_count}(\tau_{\text{XYZ}}), d \times \text{channel_count}(\tau_{\text{XYZ}}))}{(a_1) + (a_2) : d \times \text{channel_count}(\tau_{\text{XYZ}})} \text{TRIVIALBROADCAST}}{\tau_{\text{XYZ}}((a_1) + (a_2)) : (\tau_{\text{XYZ}}, d)} \text{ARRAYBINOPR} \text{INIT}$$

Since $v_1 + v_2 : (\tau_{\text{XYZ}}, d)$, $v_1 + v_2 \rightarrow \tau_{\text{XYZ}}((a_1) + (a_2))$, and $\tau_{\text{XYZ}}((a_1) + (a_2)) : (\tau_{\text{XYZ}}, d)$, preservation holds for this case.

Case 2-b: $v_1, v_2 : (\tau_{\text{LMS}}, d)$ We know from case 2-b of the progress proof for TRISTIMULUSADD that $\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{LMS}}) \mid v_1 + v_2 \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))$.

$$\frac{\frac{a_1, a_2 : d \times \text{channel_count}(\tau_{\text{LMS}}) \quad \frac{\text{broadcastable}(d \times \text{channel_count}(\tau_{\text{LMS}}), d \times \text{channel_count}(\tau_{\text{LMS}}))}{(a_1) + (a_2) : d \times \text{channel_count}(\tau_{\text{LMS}})} \text{TRIVIALBROADCAST}}{\tau_{\text{LMS}}((a_1) + (a_2)) : (\tau_{\text{LMS}}, d)} \text{ARRAYBINOPR} \text{INIT}$$

Since $v_1 + v_2 : (\tau_{\text{LMS}}, d)$, $v_1 + v_2 \rightarrow \tau_{\text{LMS}}((a_1) + (a_2))$, and $\tau_{\text{LMS}}((a_1) + (a_2)) : (\tau_{\text{LMS}}, d)$, preservation holds for this case.

Case 3-b: $v_1, v_2 : (\tau_{\text{sRGB}}, d)$ We know from case 2-c of the progress proof for TRISTIMULUSADD that $\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{sRGB}}) \mid v_1 + v_2 \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255]))$.

In order to show that $\tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])) : (\tau_{\text{sRGB}}, d)$, we must show that $(((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])) : d \times \text{channel_count}(\tau_{\text{sRGB}})$.

All operations in the above array expression are binary array operations, and can be evaluated by the rules E-ARRBINOPL and E-ARRBINOPR. These rules stipulate that the left operand must be broadcastable to the right operand, or vice versa. a_1 and a_2 both have the same dimension, $d \times \text{channel_count}(\tau_{\text{sRGB}})$. $[255]$, $\gamma_{\tau_{\text{sRGB}}}$, $\gamma_{\tau_{\text{sRGB}}}^{-1}$ all have the dimension of 1, and thus can be broadcast to $d \times \text{channel_count}(\tau_{\text{sRGB}})$ by the SCALARBROADCAST rule.

The output of the rules E-ARRBINOPL and E-ARRBINOPR has the dimension type of the dimension being broadcast to. (In $\text{broadcastable}(d_1, d_2)$, d_1 is being broadcast to d_2 .) Therefore, the final dimension type of $(((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255]))$ is $d \times \text{channel_count}(\tau_{\text{sRGB}})$.

$$\frac{((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])) : d \times \text{channel_count}(\tau_{\text{sRGB}})}{\tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])) : (\tau_{\text{sRGB}}, d)} \text{INIT}$$

Since $v_1 + v_2 : (\tau_{\text{sRGB}}, d)$, $v_1 + v_2 \rightarrow \tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255]))$, and $\tau_{\text{sRGB}}((((a_1)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) + (((a_2)/([255])) * (\gamma_{\tau_{\text{sRGB}}})) * (\gamma_{\tau_{\text{sRGB}}}^{-1})) \times ([255])) : (\tau_{\text{sRGB}}, d)$, preservation holds for this case.

Summary. Preservation holds for all cases of the TRISTIMULUSADD rule. Any non-basic type that TRISTIMULUSADD type checks can be evaluated further (PROGRESS). The type of the non-basic value is preserved after the evaluation step.

4.4.4 LightAdd Preservation

$$\frac{v_1, v_2 : (\tau_{\text{Light}}, d)}{v_1 + v_2 : (\tau_{\text{Light}}, d)} \text{LIGHTADD}$$

Case 1: $v_1 \notin v_B$. Since v_1 is not in v_B , we know from the progress theorem that there exists some v'_1 such that $v_1 \rightarrow v'_1$. Additionally, from the preservation inductive hypothesis, we know that v'_1 is of the same type as v_1 .

$$\frac{\frac{v_1 \notin v_B}{\exists v'_1 \mid v_1 \rightarrow v'_1} \text{PROGRESS}}{\exists v'_1 \mid v_1 + v_2 \rightarrow v'_1 + v_2} \text{E-PROPBINOPL}$$

$$\frac{\frac{\frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_1 : (\tau_{\text{Light}}, d)} \text{CONV-LIGHTADD} \quad v_1 \rightarrow v'_1}{v'_1 : (\tau_{\text{Light}}, d)} \text{PRES-INDHYP} \quad \frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_2 : (\tau_{\text{Light}}, d)} \text{CONV-LIGHTADD}}{v'_1 + v_2 : (\tau_{\text{Light}}, d)} \text{LIGHTADD}$$

Since $v_1 + v_2 : (\tau_{\text{Light}}, d)$, $v_1 + v_2 \rightarrow v'_1 + v_2$, and $v'_1 + v_2 : (\tau_{\text{Light}}, d)$, preservation holds for this case.

Case 2: $v_2 \notin v_B$. Since v_2 is not in v_B , we know from the progress theorem that there exists some v'_2 such that $v_2 \rightarrow v'_2$. Additionally, from the preservation inductive hypothesis, we know that v'_2 is of the same type as v_2 .

$$\frac{\frac{v_2 \notin v_B}{\exists v'_2 \mid v_2 \rightarrow v'_2} \text{PROGRESS}}{\exists v'_2 \mid v_1 + v_2 \rightarrow v_1 + v'_2} \text{E-PROPBINOPL}$$

$$\frac{\frac{\frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_2 : (\tau_{\text{Light}}, d)} \text{CONV-LIGHTADD} \quad v_2 \rightarrow v'_2}{v'_2 : (\tau_{\text{Light}}, d)} \text{PRES-INDHYP} \quad \frac{v_1 + v_2 : (\tau_{\text{Light}}, d)}{v_1 : (\tau_{\text{Light}}, d)} \text{CONV-LIGHTADD}}{v_1 + v'_2 : (\tau_{\text{Light}}, d)} \text{LIGHTADD}$$

Since $v_1 + v_2 : (\tau_{\text{Light}}, d)$, $v_1 + v_2 \rightarrow v_1 + v'_2$, and $v_1 + v'_2 : (\tau_{\text{Light}}, d)$, preservation holds for this case.

Case 3: $v_1, v_2 \notin v_B$. For this case, the proofs from both case 1 and case 2 apply. Therefore, preservation holds for this case.

Case 4: $v_1, v_2 \in v_B$. Case 2 of the progress proof for the LIGHTADD rule has the same stipulations as this case. We know from case 2 of the progress proof that $\exists a_1, a_2 : d \times \text{channel_count}(\tau_{\text{Light}}) \mid v_1 + v_2 \rightarrow \tau_{\text{Light}}((a_1) + (a_2))$.

$$\frac{\frac{a_1, a_2 : d \times \text{channel_count}(\tau_{\text{Light}}) \quad \text{broadcastable}(d \times \text{channel_count}(\tau_{\text{Light}}), d \times \text{channel_count}(\tau_{\text{Light}}))}{(a_1) + (a_2) : d \times \text{channel_count}(\tau_{\text{Light}})} \text{TRIVIALBROADCAST}}{\tau_{\text{Light}}((a_1) + (a_2)) : (\tau_{\text{Light}}, d)} \text{INIT}$$

Since $v_1 + v_2 : (\tau_{\text{Light}}, d)$, $v_1 + v_2 \rightarrow \tau_{\text{Light}}((a_1) + (a_2))$, and $\tau_{\text{Light}}((a_1) + (a_2)) : (\tau_{\text{Light}}, d)$, preservation holds for this case.

Summary. Preservation holds for all cases of the LIGHTADD rule. Any non-basic type that LIGHTADD type checks can be evaluated further (PROGRESS). The type of the non-basic value is preserved after the evaluation step.

4.4.5 Reflect Preservation

$$\frac{v_1 : (\tau_{\text{Light}}, d) \quad v_2 : (\tau_{\text{Reflectance}}, d)}{v_1 \times v_2 : (\tau_{\text{Light}}, d)} \text{REFLECT}$$

Case 1: $v_1 \notin v_B$. Since v_1 is not in v_B , we know from the progress theorem that there exists some v'_1 such that $v_1 \rightarrow v'_1$. Additionally, from the preservation inductive hypothesis, we know that v'_1 is of the same type as v_1 .

$$\frac{\frac{\frac{v_1 \notin v_B}{\exists v'_1 \mid v_1 \rightarrow v'_1} \text{PROGRESS}}{\exists v'_1 \mid v_1 \times v_2 \rightarrow v'_1 \times v_2} \text{E-PROPBINOPL}}{\frac{\frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_1 : (\tau_{\text{Light}}, d)} \text{CONV-REFLECT} \quad \frac{v_1 \rightarrow v'_1}{v'_1 : (\tau_{\text{Light}}, d)} \text{PRES-INDHYP} \quad \frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_2 : (\tau_{\text{Reflectance}}, d)} \text{CONV-REFLECT}}{v'_1 \times v_2 : (\tau_{\text{Light}}, d)} \text{REFLECT}}$$

Since $v_1 \times v_2 : (\tau_{\text{Light}}, d)$, $v_1 \times v_2 \rightarrow v'_1 \times v_2$, and $v'_1 \times v_2 : (\tau_{\text{Light}}, d)$, preservation holds for this case.

Case 2: $v_2 \notin v_B$. Since v_2 is not in v_B , we know from the progress theorem that there exists some v'_2 such that $v_2 \rightarrow v'_2$. Additionally, from the preservation inductive hypothesis, we know that v'_2 is of the same type as v_2 .

$$\frac{\frac{\frac{v_2 \notin v_B}{\exists v'_2 \mid v_2 \rightarrow v'_2} \text{PROGRESS}}{\exists v'_2 \mid v_1 \times v_2 \rightarrow v_1 \times v'_2} \text{E-PROPBINOPL}}{\frac{\frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_2 : (\tau_{\text{Reflectance}}, d)} \text{CONV-REFLECT} \quad \frac{v_2 \rightarrow v'_2}{v'_2 : (\tau_{\text{Reflectance}}, d)} \text{PRES-INDHYP} \quad \frac{v_1 \times v_2 : (\tau_{\text{Light}}, d)}{v_1 : (\tau_{\text{Light}}, d)} \text{CONV-REFLECT}}{v_1 \times v'_2 : (\tau_{\text{Light}}, d)} \text{REFLECT}}$$

Since $v_1 \times v_2 : (\tau_{\text{Light}}, d)$, $v_1 \times v_2 \rightarrow v_1 \times v'_2$, and $v_1 \times v'_2 : (\tau_{\text{Light}}, d)$, preservation holds for this case.

Case 3: $v_1, v_2 \notin v_B$. For this case, the proofs from both case 1 and case 2 apply. Therefore, preservation holds for this case.

Case 4: $v_1, v_2 \in v_B$. We know from case 3 of the progress proof that $\exists a_1 : d \times \text{channel_count}(\tau_{\text{Light}}), a_2 : d \times \text{channel_count}(\tau_{\text{Reflectance}}) \mid v_1 \times v_2 \rightarrow \tau_{\text{Light}}((a_1) \times (a_2))$.

$$\frac{\frac{\frac{\text{channel_count}(\tau_{\text{Light}}) = 89 \quad \text{channel_count}(\tau_{\text{Reflectance}}) = 89}{\text{channel_count}(\tau_{\text{Light}}) = \text{channel_count}(\tau_{\text{Reflectance}})} \text{SUBST} \quad a_2 : d \times \text{channel_count}(\tau_{\text{Reflectance}})}{a_2 : d \times \text{channel_count}(\tau_{\text{Light}})} \text{SUBST}$$

$$\frac{\frac{a_1, a_2 : d \times \text{channel_count}(\tau_{\text{Light}}) \quad \text{broadcastable}(d \times \text{channel_count}(\tau_{\text{Light}}), d \times \text{channel_count}(\tau_{\text{Light}}))}{(a_1) \times (a_2) : d \times \text{channel_count}(\tau_{\text{Light}})} \text{TRIVIALBROADCAST}}{\frac{\tau_{\text{Light}}((a_1) \times (a_2)) : (\tau_{\text{Light}}, d)}{\text{INIT}} \text{E-ARRBINOPR}}$$

Since $v_1 \times v_2 : (\tau_{\text{Light}}, d)$, $v_1 \times v_2 \rightarrow \tau_{\text{Light}}((a_1) \times (a_2))$, and $\tau_{\text{Light}}((a_1) \times (a_2)) : (\tau_{\text{Light}}, d)$, preservation holds for this case.

Summary. Preservation holds for all cases of the REFLECT rule. Any non-basic type that REFLECT type checks can be evaluated further (PROGRESS). The type of the non-basic value is preserved after the evaluation step.

4.4.6 Preservation Summary

We have shown that each SMALLERSPACE type rule fulfills the preservation theorem by the principle of induction. Any type checked SMALLERSPACE value that can take an evaluation step preserves its type after said evaluation step.

5 Equality Saturation Rewrite Rules

Tbl. 7 is a list of the rewrite rules used in our equality saturation optimizer. Part of the rules are adapted from TASO [8], which investigates rewrite rules for tensor algebra containing up to four operators. We have pruned rewrite rules that cause a substantial increase in equality saturation runtime. Rules involving the exponentiation operator (Pow) are newly introduced by us.

add-associative	$(\text{Add } x (\text{Add } y z)) \leftrightarrow (\text{Add } (\text{Add } x y) z)$
add-commutative	$(\text{Add } x y) \leftrightarrow (\text{Add } y x)$
mul-associative	$(\text{Mul } x (\text{Mul } y z)) \leftrightarrow (\text{Mul } (\text{Mul } x y) z)$
mul-commutative	$(\text{Mul } x y) \leftrightarrow (\text{Mul } y x)$
matmul-is-associative	$(\text{MatMul } x (\text{MatMul } y z)) \leftrightarrow (\text{MatMul } (\text{MatMul } x y) z)$
mul-distributes-over-add	$(\text{Mul } (\text{Add } x y) z) \leftrightarrow (\text{Add } (\text{Mul } x z) (\text{Mul } y z))$
matmul-is-linear-over-add	$(\text{MatMul } x (\text{Add } y z)) \leftrightarrow (\text{Add } (\text{MatMul } x y) (\text{MatMul } x z))$
pow-distributes-over-mul	$(\text{Pow } (\text{Mul } x y) z) \leftrightarrow (\text{Mul } (\text{Pow } x z) (\text{Pow } y z))$
div-distributes-over-add	$(\text{Div } (\text{Add } x y) z) \leftrightarrow (\text{Add } (\text{Div } x z) (\text{Div } y z))$

Table 7: Rewrite Rules

6 Benchmarking Programs in CoolerSpace

6.1 SpaceConv

The **SpaceConv** program converts an input image from sRGB space to opRGB space. COOLERSPACE performs type checking on the conversion operation (line 13) to ensure that the conversion between the input object’s type and the destination type is valid (rule CAST from Fig. 2). COOLERSPACE also abstracts away the complexity of applying and removing gamma from sRGB and opRGB. This allows programmers to focus on the semantics of manipulating an image’s color space without worrying about the implementation of color space conversion.

```
1 import coolerspace as cs
2 import sys
3
4 # Compilation Arguments
5 path = sys.argv[1]
6 shape_y = int(sys.argv[2])
7 shape_x = int(sys.argv[3])
8
9 # Inputs
10 srgb = cs.create_input("image", [shape_y, shape_x], cs.sRGB)
11
12 # Conversion from sRGB to opRGB
13 op = cs.opRGB(srgb)
14
15 # Compilation
16 cs.create_output(op)
17 cs.compile(path)
```

6.2 ColorBlindness



Figure 8: Color blindness simulation. Original image courtesy of Simon Amarasingham [5].

The **ColorBlindness** program takes as input an image in sRGB space and a projection matrix representing a specific type of dichromatic color blindness, and outputs an image simulating the effects of color blindness. The simulation algorithm is based on the the single-plane approach described by Viénot et al. [14].

Fig. 8 shows the output of the program for simulating Deuteranopia (a particular kind of “red-green blindness” due to the missing of M cones on the retina), and Tritanopia (missing S cones and thus cannot correctly perceive blue hues). As expected, red and green shades appear almost identical in the simulated Deuteranopia image; the simulated Tritanopia image correctly shows that Tritanopes cannot correctly perceive blue hues.

While most images are originally encoded in the sRGB space, principled color blindness simulation must be done in the LMS space. COOLERSPACE automatically handles the implementation logic of casting from sRGB to LMS and back. The program also demonstrates COOLERSPACE’s ability to treat colors as geometric objects and to cast them using a linear transformation matrix (Line 17). This transformation is type checked to ensure that the dimensions of the image in LMS space and the colorblindness matrix are compatible for a matrix multiplication. This type checking is performed by the TRIMATMUL rule in Fig. 2.

```
1 import coolerspace as cs
2 import sys
3
4 # Compilation Arguments
5 path = sys.argv[1]
6 shape_y = int(sys.argv[2])
7 shape_x = int(sys.argv[3])
8
9 # Inputs
10 image = cs.create_input("image", [shape_y, shape_x], cs.sRGB)
11 colorblind_matrix = cs.create_input("colorblind_matrix", [3, 3], cs.Matrix)
```

```

12
13 # Convert image to LMS
14 image_lms = cs.LMS(image)
15
16 # Apply colorblindness matrix
17 colorblind_image_lms = cs.matmul(image_lms, colorblind_matrix)
18
19 # Convert back
20 colorblind_image = cs.sRGB(colorblind_image_lms)
21
22 # Compilation
23 cs.create_output(colorblind_image)
24 cs.compile(path)

```

6.3 Adaptation



Figure 9: Chromatic adaptation simulation. Original image courtesy of Trish Hartman [7].

Adaptation is a program that simulates how the visual system adapts to the illuminant of a scene and preserves constant color perception across different illuminants [13]. Chromatic adaptation is the basis of white balancing in the camera raw processing pipeline [12]. The output of the program is shown in Fig. 9, where the original image captured under the CIE Standard Illuminant D35 (estimated) is adapted to one captured under the CIE Standard Illuminant D65 (typical daylight).

For principled chromatic adaptation, one must know the light spectra of the original and adapting illuminants, which are expressed in the `Light` type (Lines 10–11). `COOLERSPACE` facilitates color space conversion between `Light`, `LMS`, and `sRGB`. Like **ColorBlindness**, it also checks to ensure that the matrix multiplication operation on the `LMS` object in line 37 is well-formed.

```

1 import coolerspace as cs
2 import sys
3
4 # Compilation Arguments
5 path = sys.argv[1]
6 shape_y = int(sys.argv[2])
7 shape_x = int(sys.argv[3])
8
9 # Inputs
10 original_illuminant = cs.create_input("original_illuminant", [1], cs.LightSpectrum)
11 target_illuminant = cs.create_input("target_illuminant", [1], cs.LightSpectrum)
12 image = cs.create_input("image", [shape_y, shape_x], cs.sRGB)
13
14 # Convert image to LMS
15 image_lms = cs.LMS(image)
16
17 # Calculating factor to adjust lms cones by
18 original_illuminant_matrix = cs.Matrix(cs.LMS(original_illuminant))
19 target_illuminant_matrix = cs.Matrix(cs.LMS(target_illuminant))
20 abc = target_illuminant_matrix / original_illuminant_matrix
21
22 # Project to diagonal matrix
23 identity_3x3 = cs.Matrix([
24     [1, 0, 0],
25     [0, 1, 0],
26     [0, 0, 1]
27 ])
28 project_to_3x3 = cs.Matrix([

```

```

29     [1],
30     [1],
31     [1]
32 ])
33 abc_3x3 = cs.matmul(project_to_3x3, abc)
34 abc_diagonal = abc_3x3 * identity_3x3
35
36 # Apply modulation
37 modulated_image_lms = cs.matmul(image_lms, abc_diagonal)
38 modulated_image_srgb = cs.sRGB(modulated_image_lms)
39
40 # Compilation
41 cs.create_output(modulated_image_srgb)
42 cs.compile(path)

```

6.4 Interpolation

Interpolation simulates the interpolation of two sRGB images. The programmer specifies on line 14 that they want to evenly interpolate the colors of the two images. Programmers often attempt to interpolate color values directly in sRGB space [2, 1, 4, 3], which is physically incorrect, as sRGB is a non-linear color space. sRGB channels do not scale linearly with light intensity. COOLERSPACE understands that the programmer intends to mix two images evenly on line 14. COOLERSPACE performs the even interpolation in linear RGB space instead of sRGB space, thereby avoiding a potential bug.

```

1  import coolerspace as cs
2  import sys
3
4  # Compilation arguments
5  path = sys.argv[1]
6  shape_y = int(sys.argv[2])
7  shape_x = int(sys.argv[3])
8
9  # Inputs
10 image1 = cs.create_input("image1", [shape_y, shape_x], cs.sRGB)
11 image2 = cs.create_input("image2", [shape_y, shape_x], cs.sRGB)
12
13 # Interpolate between the two images by half
14 mixed = image1 * 0.5 + image2 * 0.5
15
16 # Compilation
17 cs.create_output(mixed)
18 cs.compile(path)

```

6.5 Mixing

The pigment mixing implementation in COOLERSPACE uses the Kubelka-Munk model to simulate the mixture of pigments. The Kubelka-Munk model [10, 9] relates the reflectance spectrum $R(\lambda)$ of a pigment to its scattering and absorption spectra as seen in Equ. 1, where $K(\lambda)$ and $S(\lambda)$ represent the absorption and scattering spectra, respectively, and C_i is the concentration of the i^{th} constituent pigment. The scattering and absorption spectra of a homogeneous mixture of materials can be modeled by a weighted sum [6] of the spectra of the constituent pigments. This relationship is expressed in Equ. 2.

$$R(\lambda) = 1 + \frac{K(\lambda)}{S(\lambda)} - \sqrt{\frac{K(\lambda)^2}{S(\lambda)^2} + 2\frac{K(\lambda)}{S(\lambda)}} \quad (1)$$

$$K_{mix}(\lambda) = \frac{1}{N} \sum_i^K K_i(\lambda) \times C_i, \quad S_{mix}(\lambda) = \frac{1}{N} \sum_i^K S_i(\lambda) \times C_i \quad (2)$$

Pigment mixing is a complex phenomenon to accurately model. COOLERSPACE abstracts away the complexity of the K-M model and allows the programmer to simulate pigment mixing through a single call of the `mix(.)` function. The `mix(.)` function correctly enforces that pigment mixing be done in the spectral space. The output of the mixing, `mixed`, in line 26 is still of a `Pigment Type`. We then cast `mixed` to a `Reflectance Type` (line 29) to calculate the color of the mixture (lines 31–35) under a user-input light (line 19).

```

1  import coolerspace as cs
2  import sys
3

```



```

4  # Compilation Arguments
5  path = sys.argv[1]
6  shape_y = int(sys.argv[2])
7  shape_x = int(sys.argv[3])
8
9  # Inputs
10 s1 = cs.create_input("scattering1", [shape_y, shape_x], cs.ScatteringSpectrum)
11 s2 = cs.create_input("scattering2", [shape_y, shape_x], cs.ScatteringSpectrum)
12
13 a1 = cs.create_input("absorption1", [shape_y, shape_x], cs.AbsorptionSpectrum)
14 a2 = cs.create_input("absorption2", [shape_y, shape_x], cs.AbsorptionSpectrum)
15
16 d1 = cs.create_input("density1", [shape_y, shape_x, 1], cs.Matrix)
17 d2 = cs.create_input("density2", [shape_y, shape_x, 1], cs.Matrix)
18
19 light = cs.create_input("light", [shape_y, shape_x], cs.LightSpectrum)
20
21 # Pigment creation
22 p1 = cs.Pigment(s1, a1)
23 p2 = cs.Pigment(s2, a2)
24
25 # Mix
26 mixed = cs.mix(d1, p1, d2, p2)
27
28 # Cast to reflectance
29 reflectance = cs.ReflectanceSpectrum(mixed)
30
31 # Reflect lights off pigment reflectance
32 reflected = cs.LightSpectrum(cs.Matrix(light) * cs.Matrix(reflectance))
33
34 # Convert to sRGB
35 image = cs.sRGB(reflected)
36
37 # Compilation
38 cs.create_output(image)
39 cs.compile(path)

```

6.6 LAB2HSV

The process of converting to and from LAB and HSV is extremely complicated, because both are non-linear perceptual color spaces. COOLERSPACE not only type checks the casting operation to ensure that an image expressed in LAB space can be converted to HSV space, but also handles the complex color space transformation logic behind the scenes. All the programmer has to do is express a single casting operation, as seen in line 13.

```

1  import coolerspace as cs
2  import sys
3
4  # Compilation Arguments
5  path = sys.argv[1]
6  shape_y = int(sys.argv[2])
7  shape_x = int(sys.argv[3])
8
9  # Inputs
10 lab = cs.create_input("image", [shape_y, shape_x], cs.LAB)
11
12 # Simple conversion
13 hsv = cs.HSV(lab)
14
15 # Compilation
16 cs.create_output(hsv)
17 cs.compile(path)

```

References

- [1] Answer to "adding/mixing colors in HSV Space". <https://stackoverflow.com/a/7388476>, September 2011.
- [2] Answer to "Interpolate from one color to another". <https://stackoverflow.com/a/21010385>, January 2014.
- [3] Weird interpolation between colors in hsv? <https://stackoverflow.com/q/37471461>, May 2016.

- [4] How to calculate (a physical) ratio of colors to achieve a target color? <https://math.stackexchange.com/q/4335003>, December 2021.
- [5] Simon Amarasingham. Red and green Eclectus Parrots, December 2019.
- [6] DR Duncan. The colour of pigment mixtures. *Proceedings of the Physical Society*, 52(3):390, 1940.
- [7] Trish Hartmann. Eleuthera Sunset, November 2012.
- [8] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. Taso: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 47–62, 2019.
- [9] Paul Kubelka. New contributions to the optics of intensely light-scattering materials. part i. *Josa*, 38(5):448–457, 1948.
- [10] Paul Kubelka and Franz Munk. An article on optics of paint layers. *Z. Tech. Phys*, 12(593-601):259–274, 1931.
- [11] ONNX. Open neural network exchange. <https://github.com/onnx/onnx>, 2018.
- [12] D Andrew Rowlands. Color conversion matrices in digital cameras: a tutorial. *Optical Engineering*, 59(11):110801, 2020.
- [13] Andrew Stockman and David H Brainard. Color vision mechanisms. *The Optical Society of America Handbook of Optics*, 3:11–1, 2010.
- [14] Françoise Viénot, Hans Brettel, and John D Mollon. Digital video colourmaps for checking the legibility of displays by dichromats. *Color Research & Application*, 24(4):243–252, 1999.
- [15] Günther Wyszecki and Walter Stanley Stiles. *Color science: concepts and methods, quantitative data and formulae*, volume 40. John wiley & sons, 2000.