On Stage Ordering in Staged Computation*

Zhenghao Wang and Richard R. Muntz

Computer Science Department, University of California Los Angeles

Abstract. A staged computation is a computation organized in a cascade of stages: each stage produces code for its successive stage; the final stage produces the desired output. An off-line procedure called binding time analysis (BTA) is often used to pre-convert unstaged code into staged code, i.e., code annotated with stage labels, which can guide online staged computation. For dynamic re-optimization purposes, it is advantageous for the order of stages in the cascade to change during runtime; however, the staged code may not support all permutations of stage sequences. Thus, it is both a and practical question to efficiently decide whether a specific stage sequence is valid for a staged code.

Our approach is to encode the set of valid stage sequences for a staged code off-line in a *stage ordering language* (SOL) to facilitate fast online decision. Contrary to the intuition that we only need a single generic SOL (such as the language of posets of stage labels) to sufficiently and efficiently encode the set of valid stage sequences for any staged code in any staged language, we may need different SOLs for different staged languages. We analyze several staged languages and then present a metatheory on validating a SOL for a given staged language. Our result reveals the relationship between SOLs and semantic properties of staged languages, and can influence the design of staged languages and BTA.

1 Introduction

A staged computation is a computation organized in a cascade of stages: each stage produces code for its successive stage; the final stage produces the desired output. Each stage may also be associated with an input whose value has to be known in order to carry out the computation of that stage. The major benefit of staging a computation is to reuse the intermediate code from an earlier stage when inputs of the later stages change, so as to avoid repeating part of the computation and increase run-time performance. It exists more or less explicitly in various forms in the area of programming languages, e.g., compilation, partial evaluation [4], multi-level partial evaluation [2], multi-stage programming [10,9], and run-time code generation [5,1,3].

Given an unstaged code in some (unstaged) programming language, an off-line procedure called binding-time analysis (BTA) is often used to derive a staged version with annotations of stage labels, which serve as guidelines for online

* This material is based upon work supported by the National Science Foundation under Grant Nos. 0086116, 0085773, and 9817773.

F. Pfenning and Y. Smaragdakis (Eds.): GPCE 2003, LNCS 2830, pp. 77–96, 2003.
© Springer-Verlag Berlin Heidelberg 2003

staged computation [4,2]. The staged version is called a *staged code*; corresponding to the unstaged language, the pieces of staged code form a *staged language*. It is also possible to manually write staged code [10,9].

The current state of the art BTA has a presumption on the order of stages to be followed by the online staged computation (the cascade of stages). This is even reflected in the design of staged languages, e.g., [10,9], which has a built-in assumption of a fixed stage sequence. We will argue that it is beneficial for a staged language to allow a staged code to support multiple stage sequences, and there is a need to efficiently decide whether a specific stage sequence is valid for a staged code online.

For example, consider the following staged code in the simple staged language of arithmetic expressions ($\mathcal{L}_1$):

$$(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{c} \mathsf{v_c},$$

where subscripts $\mathsf{a}, \mathsf{b}, \ldots$ are stage labels, $\mathsf{v_a}, \mathsf{v_b}, \ldots$ are placeholders for the inputs of the corresponding stages $\mathsf{a}, \mathsf{b}, \ldots$, and the informal semantics is that an annotated operator such as $-_\mathsf{b}$ means that the operation "$-$" *should and must* be carried out in stage b. It can be easily verified that $\langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$ is the only valid stage order for the above staged code:

$$\begin{aligned}(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{c} \mathsf{v_c}, 4 &\overset{\langle \mathsf{a} \rangle}{\Longrightarrow} (4 -_\mathsf{b} \mathsf{v_b}) +_\mathsf{c} \mathsf{v_c},\\ (4 -_\mathsf{b} \mathsf{v_b}) +_\mathsf{c} \mathsf{v_c}, 5 &\overset{\langle \mathsf{b} \rangle}{\Longrightarrow} (-1) +_\mathsf{c} \mathsf{v_c},\\ (-1) +_\mathsf{c} \mathsf{v_c}, 6 &\overset{\langle \mathsf{c} \rangle}{\Longrightarrow} 5.\end{aligned}$$

However, for the same semantics, the following staged code

$$(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{d} (\mathsf{v_c} \times_\mathsf{d} \mathsf{v_d}),$$

has three valid stage sequences, namely $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \rangle$, $\langle \mathsf{a}, \mathsf{c}, \mathsf{b}, \mathsf{d} \rangle$, $\langle \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{d} \rangle$.

To see why it is beneficial for a staged code to support multiple stage sequences, we need to see that the runtime efficiency of a stage sequence depends not only on the cost of each stage but also on the relative frequencies of changes to the input of each stage. Intuitively but not absolutely, one stage sequence is better than another if the input that belongs to an earlier stage changes less frequently than that of a later stage. Thus if a staged code supports multiple stage sequences, it is possible to dynamically re-optimize the stage sequence during runtime without re-doing BTA during runtime.

To facilitate efficient online decision of whether a specific stage sequence is valid for a staged code online, we propose to encode the set of valid stage sequences for a piece of staged code off-line in a *stage ordering language* (SOL). A term in a SOL denotes a set of stage sequences, and there should be an efficient algorithm to decide whether a stage sequence *conforms to* a SOL term, i.e., whether it is in the set of stage sequences denoted by a SOL term.

For example, we may use the language of partially ordered sets (posets) of stage labels as the SOL for $\mathcal{L}_1$. A stage sequence conforms to a SOL term (a

poset) if it is a topological sort of the poset. Then the SOL term for $(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{c} \mathsf{v_c}$ is $(\{\mathsf{a},\mathsf{b},\mathsf{c}\}, \sqsubseteq)$, where $\mathsf{a} \sqsubseteq \mathsf{b} \sqsubseteq \mathsf{c}$. It can be verified that the only topological sort of this poset is $\langle \mathsf{a},\mathsf{b},\mathsf{c}\rangle$. Similarly, the SOL term for $(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{d} (\mathsf{v_c} \times_\mathsf{d} \mathsf{v_d})$ is $(\{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\}, \sqsubseteq)$, where $\mathsf{a} \sqsubseteq \mathsf{b} \sqsubseteq \mathsf{d}$ and $\mathsf{c} \sqsubseteq \mathsf{d}$, as the three possible topological sorts of this poset are $\langle \mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\rangle$, $\langle \mathsf{a},\mathsf{c},\mathsf{b},\mathsf{d}\rangle$, $\langle \mathsf{c},\mathsf{a},\mathsf{b},\mathsf{d}\rangle$.

It would be nice if there were a single generic SOL that is sufficient and necessary for all staged languages. However, as we will show later, this is not possible. There do exist generic SOLs that are sufficient, like the trivial language of enumerated sets of stage sequences: $\{\langle \mathsf{a},\mathsf{b},\mathsf{c}\rangle\}$ and $\{\langle \mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\rangle, \langle \mathsf{a},\mathsf{c},\mathsf{b},\mathsf{d}\rangle, \langle \mathsf{c},\mathsf{a},\mathsf{b},\mathsf{d}\rangle\}$. However, they can be very inefficient. For example, while the size of a poset is $O(n^2)$ for n stage labels, the size of an enumerated set of stage sequences is $O(n!)$. This inefficiency is due to the fact that these generic SOLs can encode more stage sequences than possibly valid in a given staged language. For example, we will show that it is impossible for any staged code in $\mathcal{L}_1$ to have exactly the following set of valid stage sequences $\{\langle \mathsf{a},\mathsf{b},\mathsf{c}\rangle, \langle \mathsf{a},\mathsf{c},\mathsf{b}\rangle, \langle \mathsf{b},\mathsf{a},\mathsf{c}\rangle, \langle \mathsf{b},\mathsf{c},\mathsf{a}\rangle\}$, and there are far too many such sets.

In this paper, we will analyze several staged languages to show that they require different SOLs. Then, we present a metatheory that can be used to validate a SOL as both sufficient and efficient for staged languages whose semantics have certain properties[1]. The metatheory is applied to the previous several staged languages to validate their corresponding SOLs. Although we do not provide a prescriptive approach to design a SOL for a given staged language, which seems to be very hard to us, our result reveals the relationship between SOLs and semantic properties of staged languages. Our result not only helps a creative language designer to design SOLs for staged languages, but also can influence the design of staged languages and even BTA itself.

2 Related Work

In existing staged languages [6,2,9], an annotated program only describes one particular stage sequence. For example, in the 2-level language[6] used in code generation, the cascade of stages is always $\langle \mathsf{c},\mathsf{r}\rangle$, where c and r signify compile-time and run-time respectively. In MetaML[9], the nesting level of meta-brackets minus that of escapes is essentially the natural number stage label, and the cascade of stages is always $\langle 1, 2, \ldots, n\rangle$.

Nielson and Nielson have proposed to extend their 2-level language to multi-level languages [6]. They have proposed that the well-formedness rules of multi-level (types and) expressions in a particular language have characteristics that can be described by a structure on levels. Such a structure can be given in the form of a partial order [6], the signature of a many-sorted algebra [7,8], etc.

While similar, our work is different from Nielson and Nielson's multi-level languages. The most significant difference is that in the framework of multi-level languages [6,7,8], the structure on stages is an a priori property of the multi-level

[1] Our informal notion of efficiency is weaker than necessity, in that impossible stage orderings are allowed—the fewer the more efficient.

language being designed. In other words, the structure on levels is fixed for all expressions in one language, and is the basis for both the static semantics (e.g., well-formedness) and the dynamic semantics of that language. For example, a linear order such as $1 < 2 < 3 < \ldots$ makes $e_0 \overset{\langle 1 \rangle}{\Longrightarrow} e_1 \overset{\langle 2 \rangle}{\Longrightarrow} e_2 \overset{\langle 3 \rangle}{\Longrightarrow} e_3 \ldots$ the only cascading staged evaluation sequence. The language MetaML [9], as an example, has this restriction tightly built in its syntax as well as its semantics.

In contrast, we do not define a structure on stages a priori. Stage ordering is rather a property of each staged code. What kind of mathematical structure can be used to encode stage orderings is an issue whose answer depends on the semantics of the staged language under consideration. Therefore, we try to let each staged code describe as many cascades of stages as allowed by its semantics.

3 Case Study of Several Staged Languages

3.1 Some Notations and Definitions

Let us describe the semantics of staged computation with an operational semantics formalism. Suppose that we have a set of stage labels $\langle lbl \rangle$, and the metavariable s ranges over it. The "big-step" staged evaluation relation $e, input_s \overset{\langle s \rangle}{\Longrightarrow} e'$ relates a staged code e, an input to stage s $input_s$ to the outcome of this stage e'. For simplicity, we will often drop $input_s$ in the above relation: $e \overset{\langle s \rangle}{\Longrightarrow} e'$.

Given an expression e, we denote the set of stage labels that occurs in it as $S(e)$. A (finite) sequence of distinct stages $\langle s_1, \ldots, s_n \rangle$ is also denoted as $\langle s_i \rangle_{i=1..n}$. We use the metavariable σ to range over stage sequences, and use the metavariable Σ to range over sets of stage sequences. Our notation also includes: $\sigma_1 \cdot \sigma_2$ denotes the concatenation of σ_1 and σ_2; $s \in \sigma$ denotes that s occurs in σ, i.e., $\sigma = \langle \ldots, s, \ldots \rangle$; $s \to s' \prec \sigma$ denotes that $s \to s'$ is *compatible with* σ, i.e., if $s' \in \sigma$ then $\sigma = \langle \ldots, s, \ldots, s', \ldots \rangle$ or $\sigma = \langle \ldots, s = s', \ldots \rangle$. Analogous to $S(e)$, we define $S(\sigma) = \{s | s \in \sigma\}$.

Definition 1 (Valid Stage Sequence) $\langle s_i \rangle_{i=1..n}$ *is a* valid stage sequence *for* e*, denoted as* $\langle s_i \rangle_{i=1..n} \propto e$*, if there exist* $e_1, \ldots, e_n$ *s.t.*[2] $e \overset{\langle s_1 \rangle}{\Longrightarrow} e_1 \ldots \overset{\langle s_n \rangle}{\Longrightarrow} e_n$.

Definition 2 (Stage Ordering) *The* stage ordering *of* e*, denoted as* $\mathcal{O}(e)$*, is the set of all valid stage sequences for* e*, i.e.,* $\mathcal{O}(e) = \{\sigma | \sigma \propto e\}$.

Definition 3 (SOL Conformance) *A* stage ordering language *(SOL)* Ψ *is composed of a) a grammar whose set of terminals includes* $\langle lbl \rangle$*, and b) possibly a set of well-formedness rules. We use the metavariable* ψ *to vary over terms in* Ψ*. A SOL conformance relation* $\triangleright$ *is a relation between stage sequences and terms in* Ψ*. That* a stage sequence σ conforms to a SOL term ψ *is denoted as* $\sigma \triangleright \psi$*. The set of all stage sequences that conform to* ψ *is denoted as* $\mathcal{C}(\psi)$*, i.e.,* $\mathcal{C}(\psi) = \{\sigma | \sigma \triangleright \psi\}$.

[2] s.t. = such that

$$\frac{}{n', n \xRightarrow{\langle s \rangle} n'} \quad \frac{}{\mathsf{v}_s, n \xRightarrow{\langle s \rangle} n} \quad \frac{(s' \neq s)}{\mathsf{v}_{s'}, n \xRightarrow{\langle s \rangle} \mathsf{v}_{s'}} \quad \frac{e_0, n \xRightarrow{\langle s \rangle} n_0 \quad e_1, n \xRightarrow{\langle s \rangle} n_1}{e_0\ o_s\ e_1, n \xRightarrow{\langle s \rangle} f_o(n_0, n_1)} \quad \frac{e_0, n \xRightarrow{\langle s \rangle} e_0' \quad e_1, n \xRightarrow{\langle s \rangle} e_1'}{e_0\ o_{s'}\ e_1, n \xRightarrow{\langle s \rangle} e_0'\ o_{s'}\ e_1'}\ (s' \neq s)$$

Fig. 1. Formal semantics for $\mathcal{L}_1$. Here f_o is the mathematical function for o.

3.2 Staged Arithmetic Expressions $\mathcal{L}_1$

The (abstract) syntax of our first staged language $\mathcal{L}_1$ is as follows:

$$\begin{aligned} s &\in \langle lbl \rangle ::= \mathsf{a} | \mathsf{b} | \mathsf{c} | \dots \\ n &\in \langle num \rangle ::= \dots | -2 | -1 | 0 | 1 | 2 | \dots \\ o &\in \langle op \rangle ::= + | - | \times | \div \\ e &\in \langle \mathcal{L}_1 \rangle ::= n | \mathsf{v}_s | e_0\ o_s\ e_1. \end{aligned}$$

Informally, the annotated variable v_s is the placeholder for the input of stage s, i.e., input_s will substitute v_s in stage s; the annotated operator o_s means that the operation *should and must* be carried out in stage s. The formal semantics that reflects the informal "should and must" condition is shown in Fig. 1.

To obtain the stage ordering $\mathcal{O}(e)$, we note that any subexpression $e_0\ o_s\ e_1$ in e imposes a constraint on a valid stage sequence σ such that all stage label in $S(e_0) \cup S(e_1)$ should occur before or at the same position as s in σ, i.e., for each $s' \in S(e_0) \cup S(e_1)$, we have $s' \to s \prec \sigma$. Thus, by examining all subexpressions of e, we obtain a set of constraints, each of which looks like $s' \to s \prec \sigma$. $\mathcal{O}(e)$ is the set of solutions that satisfy all the constraints.

Example 4 *Let* $e = (\mathsf{v}_\mathsf{a} -_\mathsf{b} \mathsf{v}_\mathsf{b}) +_\mathsf{c} \mathsf{v}_\mathsf{c}$. *By examining all subexpressions, we obtain the following set of constraints:*

$$\mathsf{a} \to \mathsf{b} \prec \sigma \quad \mathsf{b} \to \mathsf{b} \prec \sigma \quad \mathsf{a} \to \mathsf{c} \prec \sigma \quad \mathsf{b} \to \mathsf{c} \prec \sigma \quad \mathsf{c} \to \mathsf{c} \prec \sigma.$$

The solutions are $\sigma_0 = \langle\rangle$, $\sigma_1 = \langle \mathsf{a} \rangle$, $\sigma_2 = \langle \mathsf{a}, \mathsf{b} \rangle$, *and* $\sigma_3 = \langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$, *and* $\mathcal{O}(e) = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$.

Example 5 *Let* $e = (\mathsf{v}_\mathsf{a} -_\mathsf{b} \mathsf{v}_\mathsf{b}) +_\mathsf{d} (\mathsf{v}_\mathsf{c} \times_\mathsf{d} \mathsf{v}_\mathsf{d})$. *By examining all subexpressions, we obtain the following set of constraints:*

$$\mathsf{a} \to \mathsf{b} \prec \sigma \quad \mathsf{b} \to \mathsf{b} \prec \sigma \quad \mathsf{a} \to \mathsf{d} \prec \sigma \quad \mathsf{b} \to \mathsf{d} \prec \sigma \quad \mathsf{c} \to \mathsf{d} \prec \sigma \quad \mathsf{d} \to \mathsf{d} \prec \sigma.$$

It can be solved that $\mathcal{O}(e) = \{\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \rangle, \langle \mathsf{a}, \mathsf{c}, \mathsf{b}, \mathsf{d} \rangle, \langle \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{d} \rangle, \textit{prefixes}\}$ [3].

Observing that the stage orderings of staged code in $\mathcal{L}_1$ are closely related to posets of stage labels, we can define the notion of poset conformance as follows:

Definition 6 (Poset Conformance) *Given a poset of stage labels* $\psi = (S, \sqsubseteq)$, σ conforms to ψ, *denoted as* $\sigma \triangleright \psi$, *if for each* $s, s' \in S$, $s \sqsubseteq s'$ *implies* $s \to s' \prec \sigma$.

[3] "prefixes" means prefixes of previous sequences: $\langle\rangle$, $\langle \mathsf{a} \rangle$, $\langle \mathsf{c} \rangle$, $\langle \mathsf{a}, \mathsf{b} \rangle$, $\langle \mathsf{a}, \mathsf{c} \rangle$, $\langle \mathsf{c}, \mathsf{a} \rangle$, $\langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$, $\langle \mathsf{a}, \mathsf{c}, \mathsf{b} \rangle$, $\langle \mathsf{c}, \mathsf{a}, \mathsf{b} \rangle$.

$$\frac{}{n', n \xRightarrow{\langle s \rangle} n'} \quad \frac{}{\mathsf{v}_s, n \xRightarrow{\langle s \rangle} n} \quad \frac{(s' \neq s)}{\mathsf{v}_{s'}, n \xRightarrow{\langle s \rangle} \mathsf{v}_{s'}} \quad \frac{e_0, n \xRightarrow{\langle s \rangle} n_0 \quad e_1, n \xRightarrow{\langle s \rangle} n_1}{e_0 \ o \ e_1, n \xRightarrow{\langle s \rangle} f_o(n_0, n_1)} \quad \frac{e_0, n \xRightarrow{\langle s \rangle} e_0' \quad e_1, n \xRightarrow{\langle s \rangle} e_1'}{e_0 \ o \ e_1, n \xRightarrow{\langle s \rangle} e_0' \ o \ e_1'} \text{ (otherwise)}$$

Fig. 2. Formal semantics for $\mathcal{L}_2$.

Without proving it for the time being, we conjecture that the SOL made of posets is sufficient for $\mathcal{L}_1$:

Proposition 7 (Sufficient SOL for $\mathcal{L}_1$) *For each staged code e in $\mathcal{L}_1$, there exists a poset ψ s.t. $\mathcal{C}(\psi) = \mathcal{O}(e)$.*

The poset ψ for e can be obtained by the following procedure: first make a directed graph of the constraints we have obtained for e, and then remove all stage labels involved in any cycle (except self-loops); the result is a directed acyclic graph (DAG) representing the poset ψ for e.

Example 8 *Following the above procedure, we obtain that the poset for* $(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{c} \mathsf{v_c}$ *is* $(\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \sqsubseteq)$*, where* $\mathsf{a} \sqsubseteq \mathsf{b} \sqsubseteq \mathsf{c}$*. Similarly, the poset for* $(\mathsf{v_a} -_\mathsf{b} \mathsf{v_b}) +_\mathsf{d} (\mathsf{v_c} \times_\mathsf{d} \mathsf{v_d})$ *is* $(\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}, \sqsubseteq)$*, where* $\mathsf{a} \sqsubseteq \mathsf{b} \sqsubseteq \mathsf{d}$ *and* $\mathsf{c} \sqsubseteq \mathsf{d}$.

3.3 Staged Arithmetic Expressions $\mathcal{L}_2$

$\mathcal{L}_2$ is a slight variance of $\mathcal{L}_1$:

$$e \in \langle \mathcal{L}_2 \rangle ::= n | \mathsf{v}_s | e_0 \ o \ e_1.$$

$\mathcal{L}_2$ is different from $\mathcal{L}_1$ in that operators are no longer annotated with stages[4]; accordingly, the informal semantics differs in that an operation should be carried out *as soon as* its operands are reduced to numerical values, i.e., it is not limited to a particular stage. The formal semantics that reflects the above informal "as soon as" condition is shown in Fig. 2.

Obviously, all permutations of stage sequences are valid for a given staged code in $\mathcal{L}_2$, i.e., $\mathcal{O}(e) = \{\sigma | \sigma \text{ is any stage sequence}\}$. For example, for $e = (\mathsf{v_a} - \mathsf{v_b}) + \mathsf{v_c}$, the following are all valid staged evaluation sequences:

$$e, 4 \xRightarrow{\langle \mathsf{a} \rangle} (4 - \mathsf{v_b}) + \mathsf{v_c}, 5 \xRightarrow{\langle \mathsf{b} \rangle} (-1) + \mathsf{v_c}, 6 \xRightarrow{\langle \mathsf{c} \rangle} 5;$$
$$e, 4 \xRightarrow{\langle \mathsf{a} \rangle} (4 - \mathsf{v_b}) + \mathsf{v_c}, 6 \xRightarrow{\langle \mathsf{c} \rangle} (4 - \mathsf{v_b}) + 6, 5 \xRightarrow{\langle \mathsf{b} \rangle} 5;$$
$$e, 5 \xRightarrow{\langle \mathsf{b} \rangle} (\mathsf{v_a} - 5) + \mathsf{v_c}, 4 \xRightarrow{\langle \mathsf{a} \rangle} (-1) + \mathsf{v_c}, 6 \xRightarrow{\langle \mathsf{c} \rangle} 5;$$
$$e, 5 \xRightarrow{\langle \mathsf{b} \rangle} (\mathsf{v_a} - 5) + \mathsf{v_c}, 6 \xRightarrow{\langle \mathsf{c} \rangle} (\mathsf{v_a} - 5) + 6, 4 \xRightarrow{\langle \mathsf{a} \rangle} 5;$$
$$e, 6 \xRightarrow{\langle \mathsf{c} \rangle} (\mathsf{v_a} - \mathsf{v_b}) + 6, 4 \xRightarrow{\langle \mathsf{a} \rangle} (4 - \mathsf{v_b}) + 6, 5 \xRightarrow{\langle \mathsf{b} \rangle} 5;$$
$$e, 6 \xRightarrow{\langle \mathsf{c} \rangle} (\mathsf{v_a} - \mathsf{v_b}) + 6, 5 \xRightarrow{\langle \mathsf{b} \rangle} (\mathsf{v_a} - 5) + 6, 4 \xRightarrow{\langle \mathsf{a} \rangle} 5.$$

The SOL for $\mathcal{L}_2$ contains only a dumb term $\bot$, and any σ conforms to it: $\sigma \triangleright \bot$. The following is then trivially true:

[4] Essentially, there is a vacuous BTA in this case.

$$[\text{VarN}]\ \frac{}{v \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} v} \qquad [\text{AbsN}]\ \frac{}{\lambda v.e \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} \lambda v.e}$$

$$[\text{Ap1aN}]\ \frac{e_0 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} \lambda v.e_0' \quad [e_1/v]\, e_0' \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} e'}{e_0 \,\langle s\rangle\, e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} e'}$$

$$[\text{Ap1bN}]\ \frac{e_0 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} e_0'}{e_0 \,\langle s\rangle\, e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} e_0 \,\langle s\rangle\, e_1}\ e_0' \neq \lambda v.e$$

$$[\text{Ap2N}]\ \frac{}{e_0 \,\langle s'\rangle\, e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} e_0 \,\langle s'\rangle\, e_1}\ s' \neq s$$

(a) $\overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}}$

$$[\text{VarF}]\ \frac{}{v \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} v} \qquad [\text{AbsF}]\ \frac{e \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e'}{\lambda v.e \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} \lambda v.e'}$$

$$[\text{Ap1aF}]\ \frac{e_0 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} \lambda v.e_0' \quad [e_1/v]\, e_0' \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e'}{e_0 \,\langle s\rangle\, e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e'}$$

$$[\text{Ap1bF}]\ \frac{e_0 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}} e_0' \quad e_0' \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e_0'' \quad e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e_1'}{e_0 \,\langle s\rangle\, e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e_0'' \,\langle s\rangle\, e_1'}\ e_0' \neq \lambda v.e$$

$$[\text{Ap2F}]\ \frac{e_0 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e_0' \quad e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e_1'}{e_0 \,\langle s'\rangle\, e_1 \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e_0' \,\langle s'\rangle\, e_1'}\ s' \neq s$$

(b) $\overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}}$

$$[\text{EV}]\ \frac{e \overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}} e'}{e \overset{\langle s\rangle}{\Longrightarrow} e'}\ e' \text{ is a stage-}s\text{ value}$$

(c) $\overset{\langle s\rangle}{\Longrightarrow}$

Fig. 3. Formal semantics for $\mathcal{L}_3$. $\overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{N}}$ and $\overset{\langle s\rangle}{\Longrightarrow}_{\mathbf{F}}$ are two auxiliary relations.

Proposition 9 (Sufficient SOL for $\mathcal{L}_2$) *For each staged code e in $\mathcal{L}_2$, there exists a ψ, which is $\perp$, s.t. $\mathcal{C}(\psi) = \mathcal{O}(e)$.*

3.4 $\mathcal{L}_3$-Calculus

Our third staged language $\mathcal{L}_3$-calculus is based on λ-calculus[5]:

$$s \in \langle lbl \rangle \qquad v \in \langle var \rangle \qquad e \in \langle \mathcal{L}_3 \rangle ::= v \mid \lambda v.e \mid e_0 \,\langle s\rangle\, e_1,$$

where $\langle var \rangle$ is the set of variables; the change from λ-calculus is the stage label s in the application $e_0 \,\langle s\rangle\, e_1$. The definitions of free variables ($FV(e)$), substitution ($[e'/v]\, e$), and α-equivalence ($e \equiv e'$) are analogous to those in λ-calculus.

We also define the following concepts. A *stage-s β-redex* is an expression of the form $(\lambda v.e) \,\langle s\rangle\, e'$; a *stage-$s$ normal form* is an expression without any stage-s β-redex; a *stage-s value* is an expression free of stage label s.

The informal semantics of $\mathcal{L}_3$-calculus is that all stage-s applications *s should and must* be β-reduced in stage s. The formal semantics that reflects this "should and must" condition is shown in Fig. 3[6,7]. The evaluation is normal order; it proceeds under abstractions, which reflects the essence of program generation.

Example 10 *For the expression $e = (\lambda \mathsf{x}.(\lambda \mathsf{y}.\mathsf{x} \,\langle \mathsf{a}\rangle\, \mathsf{y}) \,\langle \mathsf{b}\rangle\, \mathsf{z}) \,\langle \mathsf{a}\rangle\, (\lambda \mathsf{w}.\mathsf{w})$, we have:*

$$e \overset{\langle \mathsf{a}\rangle}{\Longrightarrow} (\lambda \mathsf{y}.\mathsf{y}) \,\langle \mathsf{b}\rangle\, \mathsf{z} \overset{\langle \mathsf{b}\rangle}{\Longrightarrow} \mathsf{z} \qquad \textit{and} \qquad e \overset{\langle \mathsf{b}\rangle}{\Longrightarrow} (\lambda \mathsf{x}.\mathsf{x} \,\langle \mathsf{a}\rangle\, \mathsf{z}) \,\langle \mathsf{a}\rangle\, (\lambda \mathsf{w}.\mathsf{w}) \overset{\langle \mathsf{a}\rangle}{\Longrightarrow} \mathsf{z}$$

[5] To disambiguate concrete syntax into the abstract syntax, we adopt the common convention that application is left associative and that e in $\lambda v.e$ extends to the first unmatched ")"or the end of the whole expression.

[6] As a result of the simplification of $\mathcal{L}_3$, no input is associated with a stage; the essence of our work is not affected.

[7] The correctness of Fig. 3 can be formalized and proved with similar ideas used to prove Church-Rosser and Standardization Theorems of λ-calculus.

Therefore $\mathcal{O}(e) = \{\langle\rangle, \langle \mathsf{a}\rangle, \langle \mathsf{b}\rangle, \langle \mathsf{a}, \mathsf{b}\rangle, \langle \mathsf{b}, \mathsf{a}\rangle\}$.

Example 11 *For the expression* $e = (\lambda \mathsf{v}.(\lambda \mathsf{x}.\mathsf{v})\ ^{\langle \mathsf{b}\rangle}\ (\mathsf{v}\ ^{\langle \mathsf{c}\rangle}\ \mathsf{w}))\ ^{\langle \mathsf{a}\rangle}\ (\lambda \mathsf{z}.\mathsf{z})$*, we have only the following valid stage sequences (and their prefixes):*

$$e \overset{\langle \mathsf{a}\rangle}{\Longrightarrow} (\lambda \mathsf{x}.\lambda \mathsf{z}.\mathsf{z})\ ^{\langle \mathsf{b}\rangle}\ ((\lambda \mathsf{z}.\mathsf{z})\ ^{\langle \mathsf{c}\rangle}\ \mathsf{w}) \overset{\langle \mathsf{b}\rangle}{\Longrightarrow} \lambda \mathsf{z}.\mathsf{z} \overset{\langle \mathsf{c}\rangle}{\Longrightarrow} \lambda \mathsf{z}.\mathsf{z}$$
$$e \overset{\langle \mathsf{a}\rangle}{\Longrightarrow} (\lambda \mathsf{x}.\lambda \mathsf{z}.\mathsf{z})\ ^{\langle \mathsf{b}\rangle}\ ((\lambda \mathsf{z}.\mathsf{z})\ ^{\langle \mathsf{c}\rangle}\ \mathsf{w}) \overset{\langle \mathsf{c}\rangle}{\Longrightarrow} (\lambda \mathsf{x}.\lambda \mathsf{z}.\mathsf{z})\ ^{\langle \mathsf{b}\rangle}\ \mathsf{w} \overset{\langle \mathsf{b}\rangle}{\Longrightarrow} \lambda \mathsf{z}.\mathsf{z}$$
$$e \overset{\langle \mathsf{b}\rangle}{\Longrightarrow} (\lambda \mathsf{v}.\mathsf{v})\ ^{\langle \mathsf{a}\rangle}\ (\lambda \mathsf{z}.\mathsf{z}) \overset{\langle \mathsf{a}\rangle}{\Longrightarrow} \lambda \mathsf{z}.\mathsf{z} \overset{\langle \mathsf{c}\rangle}{\Longrightarrow} \lambda \mathsf{z}.\mathsf{z}$$
$$e \overset{\langle \mathsf{b}\rangle}{\Longrightarrow} (\lambda \mathsf{v}.\mathsf{v})\ ^{\langle \mathsf{a}\rangle}\ (\lambda \mathsf{z}.\mathsf{z}) \overset{\langle \mathsf{c}\rangle}{\Longrightarrow} (\lambda \mathsf{v}.\mathsf{v})\ ^{\langle \mathsf{a}\rangle}\ (\lambda \mathsf{z}.\mathsf{z}) \overset{\langle \mathsf{a}\rangle}{\Longrightarrow} \lambda \mathsf{z}.\mathsf{z}$$

Therefore $\mathcal{O}(e) = \{\langle \mathsf{a}, \mathsf{b}, \mathsf{c}\rangle, \langle \mathsf{a}, \mathsf{c}, \mathsf{b}\rangle, \langle \mathsf{b}, \mathsf{a}, \mathsf{c}\rangle, \langle \mathsf{b}, \mathsf{c}, \mathsf{a}\rangle, \textit{prefixes}\}$.

The last example can be used to show that the poset SOL introduced in Sec. 3.2 is no longer sufficient for $\mathcal{L}_3$-calculus: it is impossible to find a poset ψ so that $\mathcal{C}(\psi) = \{\langle \mathsf{a}, \mathsf{b}, \mathsf{c}\rangle, \langle \mathsf{a}, \mathsf{c}, \mathsf{b}\rangle, \langle \mathsf{b}, \mathsf{a}, \mathsf{c}\rangle, \langle \mathsf{b}, \mathsf{c}, \mathsf{a}\rangle, \text{prefixes}\}$.

Finding a sufficient yet efficient SOL for $\mathcal{L}_3$-calculus turns out to be a non-trivial task; we were able to find one only after we had developed the metatheory to be discussed in Sec. 4. Let us call the SOL $\Psi_\diamond$. The grammar of $\Psi_\diamond$ is

$$\begin{aligned} &s \in \langle \mathit{lbl}\rangle \\ &S \in \langle \mathit{label\text{-}set}\rangle ::= \{s_1, \ldots, s_m\} \\ &d \in \langle \mathit{diamond}\rangle ::= S \diamond S' \\ &\psi \in \Psi_\diamond ::= \{d_1, \ldots, d_n\}. \end{aligned}$$

The intuition is that a stage ordering is described by a set of "diamonds"; each diamond is a hypercube with a starting vertex described by S and a set of edges along dimensions described by S'. The well-formedness rules are:

$$\frac{S \cap S' = \{\} \quad |S'| \geq 1}{\vdash_\diamond S \diamond S'} \qquad \frac{\vdash_\diamond d_i (i = 1..n)}{\vdash \{d_1, \ldots, d_n\}},$$

and the SOL conformance relation $\triangleright$ is defined by the following rules:

$$\text{[DMD Empty]}\ \frac{\vdash \psi}{\langle\rangle \triangleright \psi} \qquad \text{[DMD Grow1]}\ \frac{\sigma \triangleright \psi \quad S(\sigma) \diamond S \in \psi \quad s \in S}{\sigma \cdot \langle s\rangle \triangleright \psi} \tag{1}$$
$$\text{[DMD Grow2]}\ \frac{\sigma_2 \cdot \langle s\rangle \triangleright \psi \quad \sigma_2 \cdot \langle s'\rangle \triangleright \psi \quad \sigma_1 \triangleright \psi \quad S(\sigma_2 \cdot \langle s'\rangle) = S(\sigma_1) \quad s \neq s'}{\sigma_1 \cdot \langle s\rangle \triangleright \psi}.$$

We will prove later using the metatheory that $\Psi_\diamond$ is a sufficient SOL for $\mathcal{L}_3$-calculus:

Proposition 12 (Sufficient SOL for $\mathcal{L}_3$-calculus) *For each staged code e in $\mathcal{L}_3$, there exists a term $\psi \in \Psi_\diamond$ s.t. $\mathcal{C}(\psi) = \mathcal{O}(e)$.*

Example 13 *The SOL term for* $(\lambda \mathsf{x}.(\lambda \mathsf{y}.\mathsf{x}\ ^{\langle \mathsf{a}\rangle}\ \mathsf{y})\ ^{\langle \mathsf{b}\rangle}\ \mathsf{z})\ ^{\langle \mathsf{a}\rangle}\ (\lambda \mathsf{w}.\mathsf{w})$ *in Example 10, is* $\{\{\} \diamond \{\mathsf{a}, \mathsf{b}\}\}$*. Similarly, the SOL term for* $(\lambda \mathsf{v}.(\lambda \mathsf{x}.\mathsf{v})\ ^{\langle \mathsf{b}\rangle}\ (\mathsf{v}\ ^{\langle \mathsf{c}\rangle}\ \mathsf{w}))\ ^{\langle \mathsf{a}\rangle}\ (\lambda \mathsf{z}.\mathsf{z})$ *in Example 11 is* $\{\{\} \diamond \{\mathsf{a}, \mathsf{b}\}, \{\mathsf{a}\} \diamond \{\mathsf{c}\}, \{\mathsf{b}\} \diamond \{\mathsf{c}\}\}$.

4 Metatheory on Proving Sufficient SOL Propositions

In the last section, we proposed several sufficient SOL propositions for various staged languages (Propositions 7, 9, 12). In this section, we present a metatheory that can be used to prove these sufficient SOL propositions and indirectly show the efficiency of SOLs. The taste of the metatheory is like the principle of mathematical/structural/well-founded induction, in that it transforms such a proof into a more tractable procedure.

The first step is to come up with and formally prove some properties of the semantic relation $\stackrel{\langle s \rangle}{\Longrightarrow}$ that are related to stage sequences. Such a property has the flavor of the following example:

$$\text{If } e_0 \stackrel{\langle s \rangle}{\Longrightarrow} e_1 \stackrel{\langle s' \rangle}{\Longrightarrow} e_2, \text{ then there exist } e_1' \text{ and } e_2' \text{ s.t. } e_0 \stackrel{\langle s' \rangle}{\Longrightarrow} e_1' \stackrel{\langle s \rangle}{\Longrightarrow} e_2'.$$

Follow a mechanical procedure to turn these semantic properties into inference rules about valid sequences. For example, the above example property is turned into the following inference rule:

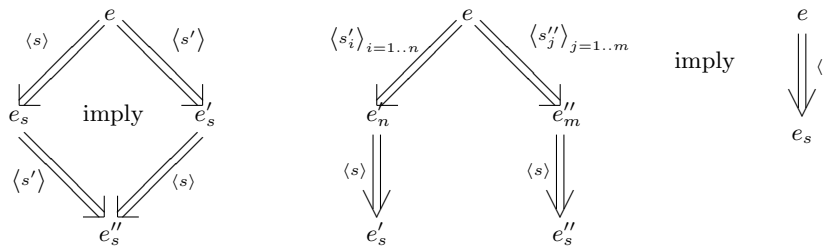
$$\frac{\sigma \cdot \langle s, s' \rangle \propto e}{\sigma \cdot \langle s', s \rangle \propto e}. \tag{2}$$

Let us denote the set of all proven semantic properties expressed in such inference rules as $\mathcal{P}$.

On the other side, specify the grammar of the target SOL Ψ formally with BNF-like grammar and, optionally, well-formedness rules. Then define the conformance relation $\sigma \triangleright \psi$ formally using inference rules. (See the example specification of $\Psi_\diamond$ in Sec. 3.4.)

The metatheory contains a meta-theorem (Theorem 20) that can be used to prove that a SOL Ψ is sufficient for a staged language, given that the semantics of the staged language satisfies properties $\mathcal{P}$. As we will see later, this meta-theorem, similar to the meta-theorem that validates well-founded induction, turns the problem into a tractable proof using standard structural/well-founded induction techniques.

The metatheory contains another meta-theorem (Theorem 21) that can be used to indirectly show that a SOL Ψ is efficient. This meta-theorem can be used to prove that the semantics of a staged language must satisfy certain properties if a SOL Ψ is sufficient for it, which is the inverse of what the other meta-theorem (Theorem 20) helps to prove. In general, the more efficient a SOL is, the less generic it is, i.e., it contains fewer stage orderings. Thus the more properties the semantics of the staged language must have if the SOL is sufficient for it. If the semantics of a staged language satisfies a property that cannot be proved using the meta-theorem w.r.t. a SOL that is sufficient for it, the SOL is probably not the most efficient SOL. Therefore, to show that a SOL is quite efficient for a staged language, we can indirectly show that many semantic properties of the staged language can be proved by the fact that the SOL is sufficient.



(a) diamond (b) branching

Fig. 4. The diamond and branching properties.

4.1 Properties of Staged Computation Semantics

First let us define some properties and check them against the semantics of the three staged languages we have introduced in Sec. 3

Definition 14 (Free Property) *For any e, s, there exists e_s s.t. $e \overset{\langle s \rangle}{\Longrightarrow} e_s$.*

Definition 15 (Diamond Property) *If $e \overset{\langle s \rangle}{\Longrightarrow} e_s$ and $e \overset{\langle s' \rangle}{\Longrightarrow} e'_s$ $(s \neq s')$, then there exists e''_s s.t. $e_s \overset{\langle s' \rangle}{\Longrightarrow} e''_s$, as illustrated in Fig. 4(a).*

Definition 16 (Branching Property) *If $e \overset{\langle s'_1 \rangle}{\Longrightarrow} \ldots \overset{\langle s'_n \rangle}{\Longrightarrow} e'_n \overset{\langle s \rangle}{\Longrightarrow} e'_s$ and $e \overset{\langle s''_1 \rangle}{\Longrightarrow} \ldots \overset{\langle s''_m \rangle}{\Longrightarrow} e''_m \overset{\langle s \rangle}{\Longrightarrow} e''_s$, where $\{s'_1, \ldots, s'_n\} \cap \{s''_1, \ldots, s''_m\} = \{\}$, then there exists e_s s.t. $e \overset{\langle s \rangle}{\Longrightarrow} e_s$, as illustrated in Fig. 4(b).*

Definition 17 (Weak Convergence Property) *If $e \overset{\langle s_{i_1} \rangle}{\Longrightarrow} e_{i_1} \overset{\langle s_{i_2} \rangle}{\Longrightarrow} \ldots \overset{\langle s_{i_n} \rangle}{\Longrightarrow} e_{i_n} \overset{\langle s_{n+1} \rangle}{\Longrightarrow} e_{i_{n+1}} \overset{\langle s_{n+2} \rangle}{\Longrightarrow} \ldots \overset{\langle s_m \rangle}{\Longrightarrow} e_{i_m}$ and $e \overset{\langle s_{j_1} \rangle}{\Longrightarrow} e_{j_1} \overset{\langle s_{j_2} \rangle}{\Longrightarrow} \ldots \overset{\langle s_{j_n} \rangle}{\Longrightarrow} e_{j_n}$, where $\langle i_1, i_2, \ldots, i_n \rangle$ and $\langle j_1, j_2, \ldots, j_n \rangle$ are permutations of $\langle 1, 2, \ldots, n \rangle$, then there exist $e_{j_{n+1}}, \ldots, e_{j_m}$ s.t. $e_{j_n} \overset{\langle s_{n+1} \rangle}{\Longrightarrow} e_{j_{n+1}} \overset{\langle s_{n+2} \rangle}{\Longrightarrow} \ldots \overset{\langle s_m \rangle}{\Longrightarrow} e_{j_m}$.*

The following two trivial properties are satisfied by all staged languages.

Definition 18 (Empty Sequence Property) $\langle\rangle \propto e$.

Definition 19 (Prefix Property) *If $\langle s_i \rangle_{i=1..n} \propto e$, then $\langle s_i \rangle_{i=1..n-1} \propto e$.*

Using standard structural induction techniques or counter-examples, we can obtain the satisfiability matrix between these properties and the semantics of the three staged languages in Sec. 3 in Table 1. Equivalent inference rules for these properties involving only $\propto e$ are also shown in the table.

Table 1. Satisfiability matrix and inference rules of some semantic properties.

	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	inference rule
Free		$\surd$		$\dfrac{\sigma \propto e}{\sigma \cdot \langle s \rangle \propto e}$
Diamond	$\surd$	$\surd$	$\surd$	$\dfrac{\sigma \cdot \langle s \rangle \propto e \quad \sigma \cdot \langle s' \rangle \propto e \quad s \neq s'}{\sigma \cdot \langle s, s' \rangle \propto e}$
Branching	$\surd$	$\surd$		$\dfrac{\sigma \cdot \sigma_1 \cdot \langle s \rangle \propto e \quad \sigma \cdot \sigma_2 \cdot \langle s \rangle \propto e \quad S(\sigma_1) \cap S(\sigma_2) = \{\}}{\sigma \cdot \langle s \rangle \propto e}$
Weak Convergence	$\surd$	$\surd$	$\surd$	$\dfrac{\sigma_1 \cdot \sigma \propto e \quad \sigma_2 \propto e \quad S(\sigma_1) = S(\sigma_2)}{\sigma_2 \cdot \sigma \propto e}$
Empty Sequence	$\surd$	$\surd$	$\surd$	$\dfrac{}{\langle\rangle \propto e}$
Prefix	$\surd$	$\surd$	$\surd$	$\dfrac{\sigma \cdot \langle s \rangle \propto e}{\sigma \propto e}$

4.2 Two Meta-theorems

Given a staged language with semantics $\stackrel{\langle s \rangle}{\Longrightarrow}$ and a SOL Ψ, we wish to relate two propositions:

- Proposition A: $\stackrel{\langle s \rangle}{\Longrightarrow}$ satisfies a set of properties $\mathcal{P}$;
- proposition B: Ψ is sufficient, i.e., for each staged code e in the stage language, there exists a term $\psi \in \Psi$ s.t. $\mathcal{C}(\psi) = \mathcal{O}(e)$.

We have the following two meta-theorems:

Theorem 20 *The "sufficient condition" part (i.e., A ⇒ B) holds, if there exists a function p that maps a set of sequences Σ to a term in Ψ: $\psi = p(\Sigma)$, s.t. a) for each Σ, $\sigma \in \Sigma$ implies $\sigma \triangleright p(\Sigma)$, and b) for each Σ that satisfies $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P})$, $\sigma \triangleright p(\Sigma)$ implies $\sigma \in \Sigma$. Here $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P})$ is the set of inference rules obtained by replacing $\propto e$ in $\mathcal{P}$ with $\in \Sigma$.*

Proof Condition a) means that $\forall \Sigma. \Sigma \subseteq \{\sigma | \sigma \triangleright p(\Sigma)\}$, while condition b) means that $\forall \Sigma$ satisfying $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P}). \{\sigma | \sigma \triangleright p(\Sigma)\} \subseteq \Sigma$. Thus

$$\exists p. \forall \Sigma \text{ satisfying } \mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P}). \{\sigma | \sigma \triangleright p(\Sigma)\} = \Sigma.$$

On the other hand, if $\propto$ is a relation that satisfies $\mathcal{P}$, then $\Sigma(e) = \{\sigma | \sigma \propto e\}$ is a set of sequences that satisfies $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P})$. Therefore we have:

$$\exists p. \forall e. \{\sigma | \sigma \triangleright p(\{\sigma | \sigma \propto e\})\} = \{\sigma | \sigma \propto e\},$$

which implies Proposition B.

□

Theorem 21 *The "necessary condition" part (i.e., B ⇒ A) holds, if all rules in $\mathcal{T}_{\propto e \to \triangleright \psi}(\mathcal{P})$ are sound w.r.t. the definition of $\triangleright$, where $\mathcal{T}_{\propto e \to \triangleright \psi}(\mathcal{P})$ is the set of inference rules obtained by replacing $\propto e$ in $\mathcal{P}$ with $\triangleright \psi$.*

Table 2. $\mathcal{T}_{\propto e \to \in \Sigma}$ and $\mathcal{T}_{\propto e \to \triangleright\psi}$ on the earlier properties.

	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{T}_{\propto e \to \in \Sigma}$	$\mathcal{T}_{\propto e \to \triangleright\psi}$
Free		√		$\frac{\sigma \in \Sigma}{\sigma \cdot \langle s \rangle \in \Sigma}$	$\frac{\sigma \triangleright \psi}{\sigma \cdot \langle s \rangle \triangleright \psi}$
Diamond	√	√	√	$\frac{\sigma \cdot \langle s \rangle \in \Sigma \;\; \sigma \cdot \langle s' \rangle \in \Sigma \;\; s \neq s'}{\sigma \cdot \langle s, s' \rangle \in \Sigma}$	$\frac{\sigma \cdot \langle s \rangle \triangleright \psi \;\; \sigma \cdot \langle s' \rangle \triangleright \psi \;\; s \neq s'}{\sigma \cdot \langle s, s' \rangle \triangleright \psi}$
Branching	√	√		$\frac{\sigma \cdot \sigma_1 \cdot \langle s \rangle \in \Sigma \;\; \sigma \cdot \sigma_2 \cdot \langle s \rangle \in \Sigma \;\; \sigma_1 \cap \sigma_2 = \{\}}{\sigma \cdot \langle s \rangle \in \Sigma}$	$\frac{\sigma \cdot \sigma_1 \cdot \langle s \rangle \triangleright \psi \;\; \sigma \cdot \sigma_2 \cdot \langle s \rangle \triangleright \psi \;\; \sigma_1 \cap \sigma_2 = \{\}}{\sigma \cdot \langle s \rangle \triangleright \psi}$
Weak Convergence	√	√	√	$\frac{\sigma_1 \cdot \sigma \in \Sigma \;\; \sigma_2 \in \Sigma \;\; S(\sigma_1) = S(\sigma_2)}{\sigma_2 \cdot \sigma \in \Sigma}$	$\frac{\sigma_1 \cdot \sigma \triangleright \psi \;\; \sigma_2 \triangleright \psi \;\; S(\sigma_1) = S(\sigma_2)}{\sigma_2 \cdot \sigma \triangleright \psi}$
Empty Sequence	√	√	√	$\frac{}{\langle\rangle \in \Sigma}$	$\frac{}{\langle\rangle \triangleright \psi}$
Prefix	√	√	√	$\frac{\sigma \cdot \langle s \rangle \in \Sigma}{\sigma \in \Sigma}$	$\frac{\sigma \cdot \langle s \rangle \triangleright \psi}{\sigma \triangleright \psi}$

Proof According to Defs. 2,3, Proposition B can be rephrased as:

$$\forall e. \exists \psi. \{\sigma | \sigma \triangleright \psi\} = \{\sigma | \sigma \propto e\},$$

which implies that, given e, we can always find $\psi(e)$ s.t. $\sigma \propto e$ iff $\sigma \triangleright \psi(e)$.

Thus, if all inference rules in $\mathcal{T}_{\propto e \to \triangleright\psi}(\mathcal{P})$ are sound, i.e., they are valid for each ψ, Proposition B implies that they must also be valid for $\psi = \psi(e)$ for each e. We can further replace $\triangleright\psi(e)$ by $\propto e$ as the two are equivalent under Proposition B; what we obtain is the original set $\mathcal{P}$ of inference rules containing $\propto e$. To summarize, if all inference rules in $\mathcal{T}_{\propto e \to \triangleright\psi}(\mathcal{P})$ are sound, then B $\Rightarrow$ A. □

For readers' convenience, we list the results of applying $\mathcal{T}_{\propto e \to \in \Sigma}$ and $\mathcal{T}_{\propto e \to \triangleright\psi}$ to the properties in Table 1 together with satisfiability matrix again in Table 2.

5 Application of Metatheory

5.1 $\mathcal{L}_1$

First, let us define the SOL $\Psi_{\sqsubseteq}$ of posets formally. Posets can be represented by *directed acyclic graphs* (DAGs)[8]. The grammar for directed graphs is

$$\begin{aligned} &s \in \langle lbl \rangle \\ &S \in \langle vertices \rangle ::= \{s_1, \ldots, s_n\} \\ &E \in \langle edges \rangle ::= \{s_1 \to s'_1, \ldots, s_m \to s'_m\} \\ &\psi \in \Psi_{\sqsubseteq} ::= (S, E). \end{aligned}$$

The well-formedness rules for DAGs are:

$$\text{[DAG Empty]} \; \frac{}{\vdash (\{\}, \{\})} \quad \text{[DAG Grow]} \; \frac{\vdash (S, E) \quad s_i \in S \; (i = 1..k) \quad s \notin S}{\vdash (S \cup \{s\}, E \cup \{s_1 \to s, \ldots, s_k \to s\})}.$$

The SOL conformance relation $\triangleright$ between stage sequences and DAGs is defined by the following "DAG conformance rules:"

[8] Note that the same poset may be represented by several different DAGs.

$$\text{[DC Empty]}\ \frac{\vdash \psi}{\langle\rangle \triangleright \psi}$$

$$\text{[DC Grow]}\ \frac{\sigma \triangleright (S,E) \quad s \notin \sigma \quad s \in S \quad \{s' | s' \to s \in E\} \subseteq S(\sigma)}{\sigma \cdot \langle s \rangle \in (S,E)}. \tag{3}$$

To show that $\Psi_{\sqsubseteq}$ is sufficient (i.e., Proposition 7), we can use Theorem 20. According to Table 2, $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P}_{\mathcal{L}_1})$ contains:

$$[\mathcal{T}_{\propto e \to \in \Sigma}\text{(Diamond)}]\ \frac{\sigma\cdot\langle s\rangle \in \Sigma \quad \sigma\cdot\langle s'\rangle \in \Sigma \quad s \neq s'}{\sigma\cdot\langle s,s'\rangle \in \Sigma}$$

$$[\mathcal{T}_{\propto e \to \in \Sigma}\text{(Branching)}]\ \frac{\sigma\cdot\sigma_1\cdot\langle s\rangle \in \Sigma \quad \sigma\cdot\sigma_2\cdot\langle s\rangle \in \Sigma \quad S(\sigma_1)\cap S(\sigma_2)=\{\}}{\sigma\cdot\langle s\rangle \in \Sigma}$$

$$[\mathcal{T}_{\propto e \to \in \Sigma}\text{(Weak Convergence)}]\ \frac{\sigma_1\cdot\sigma \in \Sigma \quad \sigma_2 \in \Sigma \quad S(\sigma_1)=S(\sigma_2)}{\sigma_2\cdot\sigma \in \Sigma}$$

$$[\mathcal{T}_{\propto e \to \in \Sigma}\text{(Empty Sequence)}]\ \frac{}{\langle\rangle \in \Sigma} \qquad [\mathcal{T}_{\propto e \to \in \Sigma}\text{(Prefix)}]\ \frac{\sigma\cdot\langle s\rangle \in \Sigma}{\sigma \in \Sigma}. \tag{4}$$

Let us define the following function p:

$$\begin{aligned} p(\Sigma) &= (S_\Sigma, E_\Sigma), \text{ where} \\ S_\Sigma &= \textstyle\bigcup_{\sigma\in\Sigma} S(\sigma), \\ E_\Sigma &= \textstyle\bigcup_{s\in S_\Sigma} \left(\bigcap_{\sigma_1\cdot\langle s\rangle\cdot\sigma_2\in\Sigma} \{s' \to s | s' \in \sigma_1\}\right). \end{aligned} \tag{5}$$

Thus, we end up to show that:

- If $\sigma \in \Sigma$, then $\sigma \triangleright p(\Sigma)$;
- Given a Σ that satisfies the rules in Eq. (4), if $\sigma \triangleright p(\Sigma)$, then $\sigma \in \Sigma$.

The proofs are shown in detail in Appendix A.1.

To show that $\Psi_{\sqsubseteq}$ is efficient, we can use Theorem 21. $\mathcal{T}_{\propto e \to \triangleright\psi}(\mathcal{P}_{\mathcal{L}_1})$ contains[9]:

$$[\mathcal{T}_{\propto e \to \triangleright\psi}\text{(Diamond)}]\ \frac{\sigma\cdot\langle s\rangle \triangleright \psi \quad \sigma\cdot\langle s'\rangle \triangleright \psi \quad s \neq s'}{\sigma\cdot\langle s,s'\rangle \triangleright \psi}$$

$$[\mathcal{T}_{\propto e \to \triangleright\psi}\text{(Branching)}]\ \frac{\sigma\cdot\sigma_1\cdot\langle s\rangle \triangleright \psi \quad \sigma\cdot\sigma_2\cdot\langle s\rangle \triangleright \psi \quad S(\sigma_1)\cap S(\sigma_2)=\{\}}{\sigma\cdot\langle s\rangle \triangleright \psi}$$

$$[\mathcal{T}_{\propto e \to \triangleright\psi}\text{(Weak Convergence)}]\ \frac{\sigma_1\cdot\sigma \triangleright \psi \quad \sigma_2 \triangleright \psi \quad S(\sigma_1)=S(\sigma_2)}{\sigma_2\cdot\sigma \triangleright \psi}$$

$$[\mathcal{T}_{\propto e \to \triangleright\psi}\text{(Empty Sequence)}]\ \frac{}{\langle\rangle \triangleright \psi}\ \text{if} \vdash \psi \qquad [\mathcal{T}_{\propto e \to \triangleright\psi}\text{(Prefix)}]\ \frac{\sigma\cdot\langle s\rangle \triangleright \psi}{\sigma \triangleright \psi}. \tag{6}$$

Thus, to show that $\Psi_{\sqsubseteq}$ enforces $\mathcal{P}_{\mathcal{L}_1}$, it is enough to show that the above rules in Eq. (6) are sound w.r.t. the definition of $\triangleright$ in Eq. (3), as detailed in Appendix A.2.

5.2 $\mathcal{L}_2$

First, the syntax of SOL $\Psi_\perp$ is trivial: $\psi \in \Psi_\perp ::= \perp$, with the trivial SOL conformance relation $\triangleright$:

$$\frac{}{\sigma \triangleright \perp}. \tag{7}$$

[9] We added a side condition ($\vdash \psi$) to the $[\mathcal{T}_{\propto e \to \triangleright\psi}\text{(Empty Sequence)}]$ rule for clarity.

To show that $\Psi_\perp$ is sufficient (i.e., Proposition 9), we can use Theorem 20. According to Table 2, $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P}_{\mathcal{L}_2})$ contains all six inference rules in the $\mathcal{T}_{\propto e \to \in \Sigma}$ column of Table 2. We define the following trivial function p:

$$p(\Sigma) = \perp.$$

Thus, we end up to show that:

- If $\sigma \in \Sigma$, then $\sigma \triangleright p(\Sigma)$;
- Given a Σ that satisfies all six rules in the $\mathcal{T}_{\propto e \to \in \Sigma}$ column of Table 2, if $\sigma \triangleright p(\Sigma)$, then $\sigma \in \Sigma$.

The first point is obvious; the second point is true because such a Σ must contain all stage sequences due to $\mathcal{T}_{\propto e \to \in \Sigma}$(Free) and $\mathcal{T}_{\propto e \to \in \Sigma}$(Empty Sequence).

To show that $\Psi_\perp$ is efficient, we can use Theorem 21. $\mathcal{T}_{\propto e \to \triangleright \psi}(\mathcal{P}_{\mathcal{L}_2})$ contains all six inference rules in the $\mathcal{T}_{\propto e \to \triangleright \psi}$ column of Table 2. Thus, to show that $\Psi_\perp$ enforces $\mathcal{P}_{\mathcal{L}_2}$, it is enough to show that the six rules in the $\mathcal{T}_{\propto e \to \triangleright \psi}$ column of Table 2 are all sound w.r.t. Eq. (7), which is obvious as $\forall \sigma. \sigma \triangleright \psi = \perp$.

5.3 $\mathcal{L}_3$

We have already defined the SOL $\Psi_\diamond$ and its conformance relation $\triangleright$ in Sec. 3.4.

To show that $\Psi_\diamond$ is sufficient (i.e., Proposition 12), we can use Theorem 20. According to Table 2, $\mathcal{T}_{\propto e \to \in \Sigma}(\mathcal{P}_{\mathcal{L}_3})$ contains:

$$[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Branching})] \ \frac{\sigma \cdot \sigma_1 \cdot \langle s \rangle \in \Sigma \quad \sigma \cdot \sigma_2 \cdot \langle s \rangle \in \Sigma \quad S(\sigma_1) \cap S(\sigma_2) = \{\}}{\sigma \cdot \langle s \rangle \in \Sigma}$$

$$[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Weak Convergence})] \ \frac{\sigma_1 \cdot \sigma \in \Sigma \quad \sigma_2 \in \Sigma \quad S(\sigma_1) = S(\sigma_2)}{\sigma_2 \cdot \sigma \in \Sigma}$$

$$[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Empty Sequence})] \ \frac{}{\langle\rangle \in \Sigma} \qquad [\mathcal{T}_{\propto e \to \in \Sigma}(\text{Prefix})] \ \frac{\sigma \cdot \langle s \rangle \in \Sigma}{\sigma \in \Sigma}. \tag{8}$$

We define the following function p:

$$p(\Sigma) = \bigcup_{\sigma \in \Sigma} \{S(\sigma_1) \diamond s | \sigma = \sigma_1 \cdot \langle s \rangle \cdot \sigma_2\}. \tag{9}$$

Thus, we end up to show that:

- If $\sigma \in \Sigma$, then $\sigma \triangleright p(\Sigma)$;
- Given a Σ that satisfies the rules in Eq. (4), if $\sigma \triangleright p(\Sigma)$, then $\sigma \in \Sigma$.

The proofs are shown in detail in Appendix B.1.

To show that $\Psi_\diamond$ is efficient, we can use Theorem 21. $\mathcal{T}_{\propto e \to \triangleright \psi}(\mathcal{P}_{\mathcal{L}_3})$ contains:

$$[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Branching})] \ \frac{\sigma \cdot \sigma_1 \cdot \langle s \rangle \triangleright \psi \quad \sigma \cdot \sigma_2 \cdot \langle s \rangle \triangleright \psi \quad S(\sigma_1) \cap S(\sigma_2) = \{\}}{\sigma \cdot \langle s \rangle \triangleright \psi}$$

$$[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Weak Convergence})] \ \frac{\sigma_1 \cdot \sigma \triangleright \psi \quad \sigma_2 \triangleright \psi \quad S(\sigma_1) = S(\sigma_2)}{\sigma_2 \cdot \sigma \triangleright \psi}$$

$$[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Empty Sequence})] \ \frac{}{\langle\rangle \triangleright \psi} \ \text{if} \vdash \psi \qquad [\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Prefix})] \ \frac{\sigma \cdot \langle s \rangle \triangleright \psi}{\sigma \triangleright \psi}. \tag{10}$$

Thus, to show that $\Psi_\diamond$ enforces $\mathcal{P}_{\mathcal{L}_3}$, it is enough to show that the above rules in Eq. (10) are sound w.r.t. the definition of $\triangleright$ in Eq. (1). The proof is shown in detail in Appendix B.2.

6 Conclusion

We have studied the problem of how to encode stage orderings of staged code in staged languages. The stage ordering language (SOL) of a staged language is key for efficient on-line re-optimization of stage sequences, because the structure of the SOL can lead to well-organized and efficient search for an optimal stage sequence. By analyzing several staged languages, we have shown that no single generic SOL exists that is both sufficient (stage orderings of all staged code can be encoded) and necessary (no redundant term exists that encodes stage ordering of no staged code) for any staged language. We have related the design and validation of a SOL with semantic properties of a staged language, by developing a metatheory about sufficient and efficient SOLs.

There are many interesting possibilities for future work. Firstly, although the metatheory provides a way to transform the validation of a SOL to simpler theorem-proving tasks, there is no prescriptive method to develop a SOL in the first place. This first step relies on human intelligence; automation or guidelines in this step are worth exploring. Secondly, we did not study the problem of normal form encoding. Both $\Psi_{\sqsubseteq}$ and $\Psi_{\diamond}$ allow multiple terms to encode the same stage ordering; it may be useful to define normal forms that uniquely represent different stage orderings. Thirdly, it is desirable that the validation phase be automated. The biggest obstacle seems to be the construction of the function p.

Beyond the technical aspects, it is even more worthwhile to explore how this work will influence the design of staged languages and even BTA itself. This is similar to the case of past research in type systems: at first type systems are installed to existing languages, but later on types become a vital basis in designing new languages. Example challenges include whether we can do BTA with a less restricted stage ordering than a linear ordering.

References

1. C. Consel and F. Noël. A general approach for run-time specialization and its application to C. In *Proc. of the 23th ACM Symposium on Principles of Programming Languages*, pages 145–156, 1996.
2. R. Glück and J. Jørgensen. Efficient multi-level generating extensions for program specialization. In *Programming Languages: Implementations, Logics and programs (PLILP'95)*, volume 982 of *LNCS*, pages 259–278. Springer, 1995.
3. B. Grant, M. Mock, M. Philipose, C. Chambers, and S. Eggers. DyC: An expressive annotation-directed dynamic compiler for C. *Theoretical Computer Science*, 248(1-2):147–199, Oct. 2000.
4. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, Englewood Cliffs, NJ, 1993.
5. P. Lee and M. Leone. Optimizing ML with run-time code generation. In *Proc. of Programming Language Design and Implementation Conf.*, pages 137–148, 1996.
6. F. Nielson and H. R. Nielson. *Two-Level Functional Languages*. Cambridge University Press, 1992.
7. F. Nielson and H. R. Nielson. *Multi-level lambda-calculi: an algebraic description*, pages 338–354. Lecture Notes in Computer Science. Springer-Verlag, 1996.

8. F. Nielson and H. R. Nielson. A prescriptive framework for designing multi-level lambda-calculi. In *Proc. of the 1997 ACM Symposium on Partial Evaluation and Semantics-based Program Manipulation*, pages 193–202, June 1997.
9. W. Taha. *Multi-Stage Programming: Its Theory and Applications.* PhD thesis, Oregon Graduate Institute of Science and Technology, Nov. 1999.
10. W. Taha and T. Sheard. Multi-stage programming with explicit annotations. In *Proc. of the 1997 ACM Symposium on Partial Evaluation and Semantics-based Program Manipulation*, pages 203–217, June 1997.

A Proofs for $\mathcal{L}_1$

A.1 The Sufficient Part

We need to show that: a) if $\sigma \in \Sigma$, then $\sigma \triangleright p(\Sigma)$, where p is defined in Eq. (5); b) given a Σ that satisfies the rules in Eq. (4), if $\sigma \triangleright p(\Sigma)$, then $\sigma \in \Sigma$.
a) We need the following lemmas.

Lemma 22 *If* $\vdash (S, E)$ *and* $E' \subseteq E$, *then* $\vdash (S, E')$.

Proof By a simple structural induction on the proofs of $\vdash (S, E)$.

Lemma 23 $\vdash p(\Sigma)$.

Proof By mathematical induction on the size of Σ.
- Base case: $\Sigma = \{\}$. $p(\Sigma) = (\{\}, \{\})$, so $\vdash p(\Sigma)$ with the [DAG Empty] rule.
- Induction step: $\Sigma = \Sigma' \cup \{\sigma\}$ $(\sigma \notin \Sigma')$. Let $p(\Sigma) = (S_\Sigma, E_\Sigma)$ and $p(\Sigma') = (S_{\Sigma'}, E_{\Sigma'})$. According to induction hypothesis, $\vdash (S_{\Sigma'}, E_{\Sigma'})$. We also define

$$E_{\Sigma',\sigma} \stackrel{\text{def}}{=} \bigcup_{s \in S_{\Sigma'}} \left(\bigcap_{\sigma_1 \cdot \langle s \rangle \cdot \sigma_2 \in \Sigma' \cup \{\sigma\}} \{s' \to s | s' \in \sigma_1\} \right).$$

  Observe $E_{\Sigma',\sigma} \subseteq E_{\Sigma'}$, so we can apply Lemma 22 and obtain $\vdash (S_{\Sigma'}, E_{\Sigma',\sigma})$. Now let us divide $\sigma = \sigma_1 \cdot \langle s_1 \rangle \cdot \sigma_2 \cdot \ldots \cdot \langle s_m \rangle \cdot \sigma_{m+1}$, where $s_1, \ldots, s_m$ are those stage labels not in $S_{\Sigma'}$. We also define S_i and $E_i (i = 0..m)$ iteratively:

$$\begin{array}{llcl} S_0 = S_{\Sigma'} & S_{i+1} & = & S_i \cup \{s_{i+1}\} \\ E_0 = E_{\Sigma',\sigma} & E_{i+1} & = & E_i \cup \{s \to s_{i+1} | s \in \sigma_1 \cdot \langle s_1 \rangle \cdot \ldots \cdot \langle s_i \rangle \cdot \sigma_{i+1}\}. \end{array}$$

  Observe that for each $s \in \sigma_1 \cdot \langle s_1 \rangle \cdot \ldots \cdot \langle s_i \rangle \cdot \sigma_{i+1}$, $s \in S_i$ and $s_{i+1} \notin S_i$, so we have if $\vdash (S_i, E_i)$, then $\vdash (S_{i+1}, E_{i+1})$, according to the [DAG Grow] rule. As we have already shown $\vdash (S_0, E_0)$, we obtain that $\vdash (S_i, E_i)\, (i = 1..m)$. Finally, it can be verified that $S_m = S_\Sigma$ and $E_m = E_\Sigma$, so that $\vdash (S_\Sigma, E_\Sigma)$. □

Now we are ready to prove that $\sigma \in \Sigma$, then $\sigma \triangleright p(\Sigma)$. Let us denote $p(\Sigma) = (S_\Sigma, E_\Sigma)$, $\sigma = \langle s_j \rangle_{j=1..k}$, and define $\sigma_i = \langle s_j \rangle_{j=1..i}$.As $s_{i+1} \notin \sigma_i$, $s_{i+1} \in S_\Sigma$, and

$$\begin{aligned} \{s' | s' \to s_{i+1} \in E_\Sigma\} &= \textstyle\bigcap_{\sigma_a \cdot \langle s_{i+1} \rangle \cdot \sigma_b \in \Sigma} \{s' | s' \in \sigma_a\} \\ &\subseteq \{s' | s' \in \sigma_a, \text{ where } \sigma_a \cdot \langle s_{i+1} \rangle \cdot \sigma_b = \sigma\} \;=\; \{s' | s' \in \sigma_i\}, \end{aligned}$$

we can apply the [DC Grow] rule to obtain that if $\sigma_i \triangleright p(\Sigma)$, then $\sigma_{i+1} \triangleright p(\Sigma)$.

On the other hand, we have $\vdash p(\Sigma)$ according to Lemma 23, and thus $\sigma_0 = \langle\rangle \triangleright p(\Sigma)$ according to the [DC Empty] rule.

Now we can obtain $\sigma_i \triangleright p(\Sigma)(i = 0..k)$. As $\sigma_k = \sigma$, we conclude that $\sigma \triangleright p(\Sigma)$.
b) We need the following lemmas. In the following, Σ satisfies the rules in Eq. (4).

Lemma 24 *If $\sigma \cdot \sigma_1 \cdot \langle s\rangle \cdot \sigma_2 \in \Sigma$ and $\sigma \cdot \langle s\rangle \in \Sigma$, then $\sigma \cdot \langle s\rangle \cdot \sigma_1 \cdot \sigma_2 \in \Sigma$.*

Proof Let us denote $\sigma_1 = \langle s_i\rangle_{i=1..l}$ and define

$$\sigma_{pk} = \sigma \cdot \langle s_i\rangle_{i=1..k} \cdot \langle s\rangle \qquad \sigma_{qk} = \sigma \cdot \langle s\rangle \cdot \langle s_i\rangle_{i=1..k} \qquad (k = 0..l)$$

We wish to show that $\sigma_{pk} \in \Sigma$ and $\sigma_{qk} \in \Sigma$ imply $\sigma_{pk+1} \in \Sigma$ and $\sigma_{qk+1} \in \Sigma$. Firstly, $\sigma \cdot \sigma_1 \cdot \langle s\rangle \cdot \sigma_2 \in \Sigma$ and the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Prefix)] rule can give us $\sigma \cdot \langle s_i\rangle_{i=1..k+1} \in \Sigma$. Together with $\sigma_{pk} \in \Sigma$, we can apply the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Diamond)] rule and obtain that $\sigma_{pk+1} \in \Sigma$ and $\sigma_{pk} \cdot \langle s_{k+1}\rangle \in \Sigma$. Then we can apply the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Weak Convergence)] rule to $\sigma_{pk} \cdot \langle s_{k+1}\rangle \in \Sigma$, $\sigma_{qk} \in \Sigma$, and $\{s | s \in \sigma_{pk}\} = \{s | s \in \sigma_{qk}\}$ and obtain that $\sigma_{qk+1} \in \Sigma$.

As $\sigma_{p0} = \sigma_{q0} = \sigma \cdot \langle s\rangle \in \Sigma$, we have $\sigma_{pk}, \sigma_{qk} \in \Sigma$. Thus $\sigma_{ql} = \sigma \cdot \langle s\rangle \cdot \sigma_1 \in \Sigma$. Together with $\sigma \cdot \sigma_1 \cdot \langle s\rangle \cdot \sigma_2 \in \Sigma$, $\{s | s \in \sigma \cdot \sigma_1 \cdot \langle s\rangle\} = \{s | s \in \sigma_{ql}\}$, and the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Weak Convergence)] rule, we conclude $\sigma_{ql} \cdot \sigma_2 = \sigma \cdot \langle s\rangle \cdot \sigma_1 \cdot \sigma_2 \in \Sigma$.

□

Let $\sigma - \sigma'$ denote a sequence obtained by removing stage labels in σ' from σ.

Corollary 25 *If $\sigma, \sigma' \in \Sigma$ and $\{s | s \in \sigma'\} \subseteq \{s | s \in \sigma\}$, then $\sigma' \cdot (\sigma - \sigma') \in \Sigma$.*

Lemma 26 *If $\sigma_1 \in \Sigma$, $\sigma_2 \in \Sigma$, $\{s | s \in \sigma_1\} \cap \{s | s \in \sigma_2\} = \{s_1, \ldots, s_l\}$, and $\sigma_1 = \sigma_{11} \cdot \langle s_1\rangle \cdot \ldots \cdot \langle s_l\rangle \cdot \sigma_{1l+1}$, then $\langle s_i\rangle_{i=1..l} \in \Sigma$.*

Proof Let us define $\sigma_{pk} = \langle s_i\rangle_{i=1..k}$ $(k = 0..l)$. We wish to obtain that if $\sigma_{pk} \in \Sigma$ then $\sigma_{pk+1} \in \Sigma$. We can apply Coro. 25 to obtain that $\sigma_{pk} \cdot (\sigma_1 - \sigma_{pk}) \in \Sigma$ and $\sigma_{pk} \cdot (\sigma_2 - \sigma_{pk}) \in \Sigma$, where $\sigma_1 - \sigma_{pk} = \sigma_{11} \cdot \ldots \cdot \sigma_{1k} \cdot \langle s_{k+1}\rangle \cdot \ldots \cdot \langle s_l\rangle \cdot \sigma_{l+1}$. Using the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Prefix)] rule, we obtain $\sigma_{pk} \cdot \sigma_{11} \cdot \ldots \cdot \sigma_{1k} \cdot \langle s_{k+1}\rangle \in \Sigma$ and $\sigma_{pk} \cdot \sigma_{2k} \cdot \langle s_{k+1}\rangle \in \Sigma$ for some σ_{2k} that satisfies $\{s | s \in \sigma_{2k} \cdot \langle s_{k+1}\rangle\} \subseteq \{s | s \in \sigma_2\}$. It can be verified that $\{s | s \in \sigma_{11} \cdot \ldots \cdot \sigma_{1k}\} \cap \{s | s \in \sigma_{2k}\} = \{\}$, so we can apply the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Branching)] rule to obtain that $\sigma_{pk+1} = \sigma_{pk} \cdot \langle s_{k+1}\rangle \in \Sigma$.

Using the [$\mathcal{T}_{\propto e \to \in \Sigma}$(Empty Sequence)] rule, we have $\sigma_{p0} = \langle\rangle \in \Sigma$. Therefore, we obtain $\sigma_{pk} \in \Sigma (k = 0..l)$. Finally $\sigma_{pl} = \langle s_i\rangle_{i=1..l} \in \Sigma$.

□

Corollary 27 *If $\sigma_k \in \Sigma (k = 1..l)$, $\bigcap_{k=1}^{l} \{s | s \in \sigma_k\} = \{s_1, \ldots, s_l\}$, and $\sigma_1 = \sigma_{11} \cdot \langle s_1\rangle \cdot \ldots \cdot \langle s_l\rangle \cdot \sigma_{1l+1}$, then $\langle s_i\rangle_{i=1..l} \in \Sigma$.*

Lemma 28 *If $\sigma \cdot \sigma_1 \in \Sigma$ and $\sigma \cdot \langle s\rangle \in \Sigma$, then $\sigma \cdot \sigma_1 \cdot \langle s\rangle \in \Sigma$.*

Proof Actually, we already proved this lemma in the proof of Lemma 24.

Lemma 29 *If $\sigma \in \Sigma$, $s \notin \sigma$, $s \in S_\Sigma$, and $\{s'|s' \to s \in E_\Sigma\} \subseteq \{s'|s' \in \sigma\}$, then $\sigma \cdot \langle s \rangle \in \Sigma$.*

Proof Let $\{\sigma_1 \cdot \langle s \rangle \cdot \sigma'_1, \ldots, \sigma_l \cdot \langle s \rangle \cdot \sigma'_l\} = \{\sigma | \sigma \in \Sigma \text{ and } s \in \sigma\}$. Using the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Prefix})]$ rule, we have $\sigma_k \cdot \langle s \rangle \in \Sigma (k = 1..l)$. It can also be shown that $\bigcap_{k=1}^{l} \{s'|s' \in \sigma_k \cdot \langle s \rangle\} = \{s'|s' \to s \in E_\Sigma\} \cup \{s\}$. Now we can apply Coro. 27 and obtain that there exists σ' s.t. $\{s'|s' \in \sigma'\} = \{s'|s' \to s \in E_\Sigma\}$ and $\sigma' \cdot \langle s \rangle \in \Sigma$.

By $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Prefix})]$ rule, $\sigma' \in \Sigma$. Together with $\sigma \in \Sigma$ and $\{s'|s' \in \sigma'\} = \{s'|s' \to s \in E_\Sigma\} \subseteq \{s'|s' \in \sigma\}$, we get $\sigma' \cdot (\sigma - \sigma') \in \Sigma$ using Coro. 25. Together with $\sigma' \cdot \langle s \rangle \in \Sigma$, we further get $\sigma' \cdot (\sigma - \sigma') \cdot \langle s \rangle$ using Lemma 28.

Finally, observe that $\{s'|s' \in \sigma' \cdot (\sigma - \sigma')\} = \{s'|s' \in \sigma\}$, we can conclude that $\sigma \cdot \langle s \rangle \in \Sigma$ using the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Weak Convergence})]$ rule.

□

Now we are ready to prove that if $\sigma \triangleright p(\Sigma)$, then $\sigma \in \Sigma$, by structural induction on the proofs of $\sigma \triangleright p(\Sigma)$.

- Case [DC Empty]. $\sigma = \langle\rangle \in \Sigma$ by the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Empty Sequence})]$ rule.
- Case [DC Grow]. Suppose $\sigma \triangleright p(\Sigma)$ is derived as:

$$\frac{\sigma_r \triangleright (S_\Sigma, E_\Sigma) \quad s \notin \sigma_r \quad s \in S_\Sigma \quad \{s'|s' \to s \in E_\Sigma\} \subseteq \{s'|s' \in \sigma_r\}}{\sigma = \sigma_r \cdot \langle s \rangle \triangleright (S_\Sigma, E_\Sigma) = p(\Sigma)}.$$

  By induction hypothesis, we have $\sigma_r \in \Sigma$. Then it is straightforward to apply Lemma 29 and obtain that $\sigma = \sigma_r \cdot \langle s \rangle \in \Sigma$.

A.2 The Efficient Part

We need to show that the five rules in Eq. (6) are sound w.r.t. the definition of $\triangleright$ in Eq. (3).

The $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Empty Sequence})]$ rule is equivalent to the [DC Empty] rule.

The $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Prefix})]$ rule can be proved by a straightforward structural induction on the proofs of $\sigma \cdot \langle s \rangle \triangleright \psi$.

The $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Weak Convergence})]$ rule can be proved by a mathematical induction on the lengths of σ; the induction step needs the $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Prefix})]$ rule and the following lemma:

If $\sigma_1 \cdot \{s\} \triangleright \psi$, $\sigma_2 \triangleright \psi$, and $\{s|s \in \sigma_1\} = \{s|s \in \sigma_2\}$, then $\sigma_2 \cdot \{s\} \triangleright \psi$,
which can be proved by by structural induction on the proofs of $\sigma_1 \cdot \{s\} \triangleright \psi$:

- Case [DC Empty]. $\sigma_1 \cdot \langle s \rangle \triangleright \psi$ cannot be derived by this rule.
- Case [DC Grow]. Suppose $\sigma \cdot \langle s \rangle \triangleright \psi$ is derived as:

$$\frac{\sigma_1 \triangleright (S, E) \quad s \notin \sigma_1 \quad s \in S \quad \{s'|s' \to s \in E\} \subseteq \{s'|s' \in \sigma_1\}}{\sigma_1 \cdot \langle s \rangle \in (S, E) = \psi}.$$

  As $\{s|s \in \sigma_1\} = \{s|s \in \sigma_2\}$, $s \notin \sigma_1$ and $\{s'|s' \to s \in E\} \subseteq \{s'|s' \in \sigma_1\}$ is equivalent to $s \notin \sigma_2$ and $\{s'|s' \to s \in E\} \subseteq \{s'|s' \in \sigma_2\}$ respectively. In addition, we have $\sigma_2 = \psi = (S, E)$ and $s \in S$. Now we can apply the [DC Grow] rule and obtain that $\sigma_2 \cdot \langle s \rangle \triangleright (S, E) = \psi$.

The $[\mathcal{T}_{\propto e \to \triangleright\psi}(\text{Diamond})]$ rule can be proved by a structural induction on the proofs of $\sigma \cdot \langle s' \rangle \triangleright \psi$:

- Case [DC Empty]. $\sigma \cdot \langle s' \rangle \triangleright \psi$ cannot be derived by this rule.
- Case [DC Grow]. Suppose $\sigma \cdot \langle s' \rangle \triangleright \psi$ is derived as:

$$\frac{\sigma \triangleright (S, E) \quad s' \notin \sigma \quad s' \in S \quad \{s_r | s_r \to s' \in E\} \subseteq \{s_r | s_r \in \sigma\}}{\sigma \cdot \langle s' \rangle \in (S, E) = \psi}.$$

From the premises of the $[\mathcal{T}_{\propto e \to \triangleright\psi}(\text{Diamond})]$ rule, we have $\sigma \cdot \langle s \rangle \triangleright \psi = (S, E)$ and $s \neq s'$. $s' \notin \sigma$ and $s \neq s'$ imply $s' \notin \sigma \cdot \langle s \rangle$. $\{s_r | s_r \to s' \in E\} \subseteq \{s_r | s_r \in \sigma\} \subset \{s_r | s_r \in \sigma \cdot \langle s \rangle\}$. Now we can apply the [DC Grow] rule and obtain that $\sigma \cdot \langle s, s' \rangle = (\sigma \cdot \langle s \rangle) \cdot \langle s' \rangle \triangleright (S, E) = \psi$.

The $[\mathcal{T}_{\propto e \to \triangleright\psi}(\text{Branching})]$ rule can be proved by a structural induction on the proofs of $\sigma \cdot \sigma_1 \cdot \langle s \rangle \triangleright \psi$:

- Case [DC Empty]. $\sigma \cdot \sigma_1 \cdot \langle s \rangle \triangleright \psi$ cannot be derived by this rule.
- Case [DC Grow]. Suppose $\sigma \cdot \sigma_1 \cdot \langle s \rangle \triangleright \psi$ is derived as:

$$\frac{\sigma \cdot \sigma_1 \triangleright (S, E) \quad s \notin \sigma \cdot \sigma_1 \quad s \in S \quad \{s' | s' \to s \in E\} \subseteq \{s' | s' \in \sigma \cdot \sigma_1\}}{\sigma \cdot \sigma_1 \cdot \langle s' \rangle \in (S, E) = \psi}.$$

As $\sigma \cdot \sigma_1 \triangleright (S, E)$, we have $\sigma \triangleright (S, E)$ according the $[\mathcal{T}_{\propto e \to \triangleright\psi}(\text{Diamond})]$ rule. $s \notin \sigma \cdot \sigma_1$ implies $s \notin \sigma$.

From the premises of the $[\mathcal{T}_{\propto e \to \triangleright\psi}(\text{Branching})]$ rule, we have $\sigma \cdot \sigma_2 \cdot \langle s \rangle \triangleright \psi = (S, E)$, which can only be derived by the [DC Grow] rule as:

$$\frac{\sigma \cdot \sigma_2 \triangleright (S, E) \quad s \notin \sigma \cdot \sigma_2 \quad s \in S \quad \{s' | s' \to s \in E\} \subseteq \{s' | s' \in \sigma \cdot \sigma_2\}}{\sigma \cdot \sigma_2 \cdot \langle s' \rangle \in (S, E) = \psi}.$$

Thus $\{s' | s' \to s \in E\} \subseteq \{s' | s' \in \sigma \cdot \sigma_2\}$. Together with $\{s' | s' \to s \in E\} \subseteq \{s' | s' \in \sigma \cdot \sigma_1\}$, we obtain that $\{s' | s' \to s \in E\} \subseteq \sigma \cdot \sigma_1 \cap \sigma \cdot \sigma_2$, where we have abused the notation of $\sigma \cdot \sigma_1 \cap \sigma \cdot \sigma_2$ to really mean $\{s | s \in \sigma \cdot \sigma_1\} \cap \{s | s \in \sigma \cdot \sigma_2\}$. Again from the premises of the $[\mathcal{T}_{\propto e \to \triangleright\psi}(\text{Branching})]$ rule, $\sigma_1 \cap \sigma_2 = \{\}$, so $\sigma \cdot \sigma_1 \cap \sigma \cdot \sigma_2 = \sigma$. Thus we have $\{s' | s' \to s \in E\} \subseteq \{s' \in \sigma\}$. Now we can apply the [DC Grow] rule and obtain that $\sigma \cdot \langle s \rangle \triangleright (S, E) = \psi$.

B Proofs for $\mathcal{L}_3$

B.1 The Sufficient Part

a) We can show the following lemmas by structural induction on the appropriate structure in each lemma:

- $\vdash p(\Sigma)$;
- $\sigma \triangleright \{S(\sigma_1) \diamond s | \sigma = \sigma_1 \cdot \langle s \rangle \cdot \sigma_2\}$;
- If $\sigma \triangleright \psi$, $\vdash \psi'$, and $\psi \subseteq \psi'$, then $\sigma \triangleright \psi'$.

Then we can show assertion a) holds easily.

b) By structural induction on the proofs of $\sigma \triangleright p(\Sigma)$:

- Case [DMD Empty]. $\sigma = \langle\rangle \in \Sigma$ by the $\mathcal{T}_{\propto e \to \in \Sigma}(\text{Empty Sequence})$ rule.

– Case [DMD Grow1]. Suppose $\sigma \triangleright p(\Sigma)$ is derived as:

$$\frac{\sigma_r \triangleright p(\Sigma) \quad S(\sigma_r) \diamond S \in p(\Sigma) \quad s \in S}{\sigma = \sigma_r \cdot \langle s \rangle \triangleright p(\Sigma)}.$$

According to Eq. (9), $S(\sigma_r) \diamond S \in p(\Sigma)$ means that there exists $\sigma_p \in \Sigma$ s.t. $S(\sigma_r) \diamond S \in \{S(\sigma_{p1}) \diamond s | \sigma_p = \sigma_{p1} \cdot \langle s \rangle \cdot \sigma_{p2}\}$. As $s \in S$, this means there exists $\sigma_{p1} \cdot \langle s \rangle \cdot \sigma_{p2} \in \Sigma$, s.t. $\{s | s \in \sigma_{p1}\} = \{s | s \in \sigma_r\}$. From $\sigma_{p1} \cdot \langle s \rangle \cdot \sigma_{p2} \in \Sigma$, we can obtain $\sigma_{p1} \cdot \langle s \rangle \in \Sigma$ using the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Prefix})]$ rule. From $\sigma_r \triangleright p(\Sigma)$, we can obtain $\sigma_r \in \Sigma$ by induction hypothesis. Now we can apply the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Weak Convergence})]$ rule to $\sigma_{p1} \cdot \langle s \rangle \in \Sigma$, $\sigma_r \in \Sigma$, and $\{s | s \in \sigma_{p1}\} = \{s | s \in \sigma_r\}$ and obtain that $\sigma = \sigma_r \cdot \langle s \rangle \in \Sigma$.

– Case [DMD Grow2]. Suppose $\sigma \triangleright p(\Sigma)$ is derived as:

$$\frac{\sigma_2 \cdot \langle s \rangle \triangleright p(\Sigma) \quad \sigma_2 \cdot \langle s' \rangle \triangleright p(\Sigma) \quad \sigma_1 \triangleright p(\Sigma) \quad \{s | s \in \sigma_2 \cdot \langle s' \rangle\} = \{s | s \in \sigma_1\}}{\sigma = \sigma_1 \cdot \langle s \rangle \triangleright p(\Sigma)}.$$

By applying the induction hypothesis to $\sigma_2 \cdot \langle s \rangle \triangleright p(\Sigma)$, $\sigma_2 \cdot \langle s' \rangle \triangleright p(\Sigma)$, and $\sigma_1 \triangleright p(\Sigma)$, we obtain $\sigma_2 \cdot \langle s \rangle \in \Sigma$, $\sigma_2 \cdot \langle s' \rangle \in \Sigma$, and $\sigma_1 \in \Sigma$. Using the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Diamond})]$ rule, $\sigma_2 \cdot \langle s \rangle \in \Sigma$ and $\sigma_2 \cdot \langle s' \rangle \in \Sigma$ imply $\sigma_2 \cdot \langle s', s \rangle \in \Sigma$. Together with $\sigma_1 \in \Sigma$ and $\{s | s \in \sigma_2 \cdot \langle s' \rangle\} = \{s | s \in \sigma_1\}$, we can apply the $[\mathcal{T}_{\propto e \to \in \Sigma}(\text{Weak Convergence})]$ rule and obtain that $\sigma = \sigma_1 \cdot \langle s \rangle \in \Sigma$.

B.2 The Efficient Part

The $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Empty Sequence})]$ rule is equivalent to the [DMD Empty] rule; the $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Prefix})]$ rule can be proved by a straightforward structural induction on the proofs of $\sigma \cdot \langle s \rangle \triangleright \psi$; the $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Diamond})]$ rule is a corollary of the [DMD Grow2] rule. The $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Weak Convergence})]$ rule is the interesting one. It can be proved by mathematical induction on the lengths of σ. The induction step needs the $[\mathcal{T}_{\propto e \to \triangleright \psi}(\text{Prefix})]$ rule and the following lemma:

If $\sigma_1 \cdot \langle s \rangle \triangleright \psi$, $\sigma_2 \triangleright \psi$, and $\{s | s \in \sigma_1\} = \{s | s \in \sigma_2\}$, then $\sigma_2 \cdot \langle s \rangle \triangleright \psi$,

which can be proved by structural induction on the proofs of $\sigma_1 \cdot \langle s \rangle \triangleright \psi$:

– Case [DMD Empty]. $\sigma_1 \cdot \langle s \rangle \triangleright \psi$ cannot be derived by this rule.
– Case [DMD Grow1]. Suppose $\sigma_1 \cdot \langle s \rangle \triangleright \psi$ is derived as:

$$\frac{\sigma_1 \triangleright \psi \quad \{s | s \in \sigma_1\} \diamond S \in \psi \quad s \in S}{\sigma_1 \cdot \langle s \rangle \triangleright \psi}$$

$\{s | s \in \sigma_1\} = \{s | s \in \sigma_2\}$ and $\{s | s \in \sigma_1\} \diamond S \in \psi$ imply $\{s | s \in \sigma_2\} \diamond S \in \psi$. Together with $\sigma_2 \triangleright \psi$, $s \in S$ and the [DMD Grow2] rule, we obtain $\sigma_2 \cdot \langle s \rangle \triangleright \psi$.
– Case [DMD Grow2]. Similar to case [DMD Grow1].