Rule Formats for Structural Operational Semantics

A Very Short Introduction

Luca Aceto
ICE-TCS, School of Computer Science, Reykjavik University

Chinese Academy of Sciences, Beijing, 21 November 2013

I have worked on rule formats for SOS with many co-authors and thank them all!

Message and Overview of the Talk

Message in a Nutshell

Rules rule! Using rule formats one can prove semantic properties for classes of languages by purely syntactic means.

Overview

- Motivation for developing rule formats
- Introduction to rule formats with examples
- Tailoring a rule format (unit elements)
- Conclusion: Further and future work

The General Setting

Fact of (Computer Science) Life

In Computer Science, we use formal languages to communicate with machines (programming languages) and describe expected properties of computations (specification languages).

Like natural languages, the languages we use have

1. a syntax and
2. a semantics.

Question

How are those described in CS?

The General Setting

Fact of (Computer Science) Life

In Computer Science, we use formal languages to communicate with machines (programming languages) and describe expected properties of computations (specification languages).

Like natural languages, the languages we use have

1. a syntax and
2. a semantics.

Question

How are those described in CS?

Description of Programming and Specification Languages I

Syntax

Formally specified using, e.g., BNF notation. Example?

CCS	nil $\mathbf{0}$	prefixing at
	choice $t + u$	parallel $t \| u$

where a is an action drawn from a non-empty set A.

Benefits: Too many to mention! For instance, compiler technology was revolutionized and went from art to science, in the sense of Knuth.

State of Play (Syntax)

The syntax of every language under the sky is formally specified.

Uncontroversial!

Semantics: Logic at Work

Operational Semantics: What the program does

Meaning of a program $\approx$ Execution on an idealized machine. How is this specified?

Plotkin's answer: Use logic! Define the semantics by using inference rules.

Structural Operational Semantics for CCS

Given by transitions between terms of the form $t \xrightarrow{a} u$. These associate a loop-free finite automaton with each term. How?

$$\frac{}{ax \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{\bar{a}} y'}{x \| y \xrightarrow{\tau} x' \| y'}$$

Semantics: Logic at Work

Operational Semantics: What the program does

Meaning of a program $\approx$ Execution on an idealized machine. How is this specified?

Plotkin's answer: Use logic! Define the semantics by using inference rules.

Structural Operational Semantics for CCS

Given by transitions between terms of the form $t \xrightarrow{a} u$. These associate a loop-free finite automaton with each term. How?

$$\frac{}{ax \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{x \xrightarrow{a} x',\ y \xrightarrow{\bar{a}} y'}{x \| y \xrightarrow{\tau} x' \| y'}$$

From SOS to Properties of Languages

Pay-off: Using the SOS rules one can prove properties of CCS!

- Bisimilarity is a congruence.
- $+$ and $\|$ are commutative and associative modulo bisimilarity.
- $+$ is an idempotent binary operation—that is, $x + x = x$ holds modulo bisimilarity.
- $\mathbf{0}$ is a left and right unit for $+$ and $\|$—for example, $\mathbf{0} + x = x$ holds modulo bisimilarity.
- The idempotent, commutative monoid laws axiomatize bisimilarity over the "language of synchronization trees".
- Each program has an associated finite transition system.

Summary of the State of Play

A posteriori verification

Give the semantics and then use it to prove that the language properties one desires are valid in the semantic model.

Cons

- This is a lot of work that is repeated for many languages.
- The work needs to be redone if we modify the inference rules and/or add new operators.
- Redoing the work does not give much insight as to why the properties hold at all.

Can one do things differently?

Summary of the State of Play

A posteriori verification

Give the semantics and then use it to prove that the language properties one desires are valid in the semantic model.

Cons

- This is a lot of work that is repeated for many languages.
- The work needs to be redone if we modify the inference rules and/or add new operators.
- Redoing the work does not give much insight as to why the properties hold at all.

Can one do things differently?

An Alternative Approach: SOS Rule Formats

"Intelligent design"

Give syntactic templates for the inference rules used in defining the operational semantics for certain operators that guarantee the validity of the laws by design!

Why is this good?

- Rule formats give sufficient syntactic conditions guaranteeing semantic properties.
- Checking semantic properties can be done via syntactic checks.

An Alternative Approach: SOS Rule Formats

"Intelligent design"

Give syntactic templates for the inference rules used in defining the operational semantics for certain operators that guarantee the validity of the laws by design!

Why is this good?

- Rule formats give sufficient syntactic conditions guaranteeing semantic properties.
- Checking semantic properties can be done via syntactic checks.

Exhibit 1

Example: Ensuring that Bisimilarity is a Congruence

Write the SOS rules for your language in the following form, due to Groote and Vaandrager, and you are done!

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t}$$

where the x's and the y's are different variables, and the t's are terms.

Why should the variables be different?

Consider the rule

$$\frac{}{f(x, x) \xrightarrow{a} \mathbf{0}} \ .$$

Then $\mathbf{0}$ and $\mathbf{0} + \mathbf{0}$ have the same behaviour, but $f(\mathbf{0}, \mathbf{0})$ and $f(\mathbf{0}, \mathbf{0} + \mathbf{0})$ do not. Syntactic pattern matching is bad!

Exhibit 1

Example: Ensuring that Bisimilarity is a Congruence

Write the SOS rules for your language in the following form, due to Groote and Vaandrager, and you are done!

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t}$$

where the x's and the y's are different variables, and the t's are terms.

Why should the variables be different?

Consider the rule

$$\frac{}{f(x, x) \xrightarrow{a} \mathbf{0}} \ .$$

Then $\mathbf{0}$ and $\mathbf{0} + \mathbf{0}$ have the same behaviour, but $f(\mathbf{0}, \mathbf{0})$ and $f(\mathbf{0}, \mathbf{0} + \mathbf{0})$ do not. Syntactic pattern matching is bad!

Exhibit 2

Example: Ensuring Finite Branching

Write a finite number of SOS rules for your language in the following form (due to Bloom, Istrail and Meyer) and you are done!

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \overset{b_{ik}}{\not\rightarrow} \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\vec{x}) \xrightarrow{a} C[\vec{x}, \vec{y}]}$$

Example: A finite number of rules is not enough

$$\frac{}{f(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{f(x) \xrightarrow{a} y}{f(x) \xrightarrow{a} ay}$$

Then $f(\mathbf{0}) \xrightarrow{a} a^n\mathbf{0}$ for each $n \geq 0$.

Exhibit 2

Example: Ensuring Finite Branching

Write a finite number of SOS rules for your language in the following form (due to Bloom, Istrail and Meyer) and you are done!

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \overset{b_{ik}}{\not\rightarrow} \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\vec{x}) \xrightarrow{a} C[\vec{x}, \vec{y}]}$$

Example: A finite number of rules is not enough

$$\frac{}{f(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{f(x) \xrightarrow{a} y}{f(x) \xrightarrow{a} ay}$$

Then $f(\mathbf{0}) \xrightarrow{a} a^n\mathbf{0}$ for each $n \geq 0$.

Purposes of "Intelligent Design" and Challenges

1. Rule formats can be used to prove properties for classes of languages in one fell swoop.
2. Rule formats pave the way for a tool-set that can mechanically prove semantic properties without involving user interaction.
3. Rule formats may serve as a guideline for language designers who want to ensure, a priori, that the languages under design enjoy certain basic semantic properties.
4. Rule formats highlight the link between rule templates and semantic properties of languages.

Trade-off: Generality vs. ease of application. Logic is an experimental science! (The 80-20 rule applies.)

Tailoring a Rule Format for Unit Elements

Rest of This Talk: A Rule Format for Unit Elements

A rule format guaranteeing that certain constants are left- or right-unit elements for a set of binary operators.

Technical Question: How can we guarantee the validity of equations like

$$f(c, x) = x \quad \text{and } f(x, c) = x?$$

Let's tailor a custom-made rule format!

Intuition

Assumption: For each constant c, we assume that each c-defining inference rule is an axiom of the form $c \xrightarrow{l} p$ for some label l and term p without variables.

Question: When is c a left unit for a binary operation f?

Wish List

1. Desideratum 1: $f(c, p)$ should be able to mimic the behaviour of p for each program p.
2. Desideratum 2: $f(c, p)$ can only mimic the behaviour of p.

How do we ensure those properties syntactically in a way that allows us to handle examples from the literature (and more)?

Intuition

Assumption: For each constant c, we assume that each c-defining inference rule is an axiom of the form $c \xrightarrow{l} p$ for some label l and term p without variables.

Question: When is c a left unit for a binary operation f?

Wish List

1. Desideratum 1: $f(c, p)$ should be able to mimic the behaviour of p for each program p.
2. Desideratum 2: $f(c, p)$ can only mimic the behaviour of p.

How do we ensure those properties syntactically in a way that allows us to handle examples from the literature (and more)?

Two Examples with a Message: Part I

Example 1

Consider the binary operators f_i, $i \geq 0$, with rules

$$\frac{x_1 \xrightarrow{a} y_1}{f_i(x_0, x_1) \xrightarrow{a} f_{i+1}(x_0, y_1)} \ .$$

Any program is a left unit for each f_i. Why?
Lesson 1: We may need to consider possibly infinite sets of operators coinductively.

Two Examples with a Message: Part II

Example 2

Consider the following operations.

$$\frac{y \xrightarrow{a} y'}{f(x,y) \xrightarrow{a} g(y',x)} \qquad \frac{x \xrightarrow{a} x'}{g(x,y) \xrightarrow{a} f(y,x')}$$

Any program is a left unit for f and a right unit for g. Why?
Lesson 2: We may need to consider left and right units at the same time.

The Rule Format: Part I

Given a "language specification", the sets L and R of pairs of binary function symbols and constants are the largest sets satisfying the following constraints.

Constraint 1.L (Implements: $f(c,p)$ should be able to mimic the behaviour of p for each program p.) For each $(f,c) \in L$ and each action label a, there exists at least one rule of the following form:

$$\frac{\{x_0 \xrightarrow{a_i} y_i \mid i \in I\} \cup \{x_0 \xrightarrow{a_j} \mid j \in J\} \cup \{x_1 \xrightarrow{a} y_1\}}{f(x_0, x_1) \xrightarrow{a} t'},$$

where (1) the variables are all pairwise distinct, (2) the axioms for c "satisfy the premises involving x_0" and (3) "y_1 can be proved equal to a suitable instantiation of t'" using the laws $f(e,x) = x$ when $(f,e) \in L$ and $g(x,e') = x$ when $(g,e') \in R$.

The Rule Format: Part II

Constraint 2.L (Implements: $f(c, p)$ can only mimic the behaviour of p.) Each f-defining rule has the following form:

$$\frac{\Phi}{f(t_0, t_1) \xrightarrow{a} t'}$$

where, for each closed substitution σ such that $\sigma(t_0) \equiv c$,

1. either there exists some $t_1 \xrightarrow{a} t'' \in \Phi$ with $\sigma(t')$ "behaving like" $\sigma(t'')$, or
2. there exists a premise $\phi \in \Phi$ with t_0 as its source that cannot be met by c.

The constraints for R are symmetric.

The Syntactic Constraints At Work

Main Theorem

Consider a language with "operational semantics given by inference rules". Assume that L and R are the sets defined as given previously. For each $(f, c) \in L$, the equation $f(c, x) = x$ is valid "up to any reasonable notion of equivalence over programs". Symmetrically, for each $(f, c) \in R$, the equation $f(x, c) = x$ is valid "up to any reasonable notion of equivalence over programs".

The proof relies on the construction of suitable bisimulations.

Cool, but can it be (easily) applied to examples from the literature?

The Syntactic Constraints At Work

Main Theorem

Consider a language with "operational semantics given by inference rules". Assume that L and R are the sets defined as given previously. For each $(f, c) \in L$, the equation $f(c, x) = x$ is valid "up to any reasonable notion of equivalence over programs". Symmetrically, for each $(f, c) \in R$, the equation $f(x, c) = x$ is valid "up to any reasonable notion of equivalence over programs".

The proof relies on the construction of suitable bisimulations.
Cool, but can it be (easily) applied to examples from the literature?

Examples I

Nondeterministic Choice

Recall the rules for CCS $+$:

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

The sets $R = L = \{(+, \mathbf{0})\}$ meet the constraints. Therefore, our theorem yields the soundness of the well known equations:

$$\mathbf{0} + x = x = x + \mathbf{0}.$$

Examples II

Synchronous Parallel Composition (Hoare's CSP)

$$\frac{}{\mathrm{RUN}_a \xrightarrow{a} \mathrm{RUN}_a} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_a y \xrightarrow{a} x' \parallel_a y'} \ .$$

The sets $L = R = \{(\parallel_a, \mathrm{RUN}_a)\}$ meet the constraints. Therefore, our theorem yields the soundness of the well known equations:

$$\mathrm{RUN}_a \parallel_a x = x = x \parallel_a \mathrm{RUN}_a.$$

Examples III

Left Merge and Interleaving Parallel Composition

The following rules describe the operational semantics of the classic left merge and interleaving parallel composition operators.

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y} \qquad \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$$

The sets $L = \{(\parallel, \mathbf{0})\}$ and $R = \{(\parallel, \mathbf{0}), (\between, \mathbf{0})\}$ meet the constraints. Therefore, our theorem yields the soundness of the well known equations:

$$\mathbf{0} \parallel x = x, \quad x \parallel \mathbf{0} = x \quad \text{and} \quad x \between \mathbf{0} = x \ .$$

Note that the pair $(\between, \mathbf{0})$ cannot be added to L while preserving the constraints! Indeed, $\mathbf{0}$ is not a left unit for the left merge operator

Examples III

Left Merge and Interleaving Parallel Composition

The following rules describe the operational semantics of the classic left merge and interleaving parallel composition operators.

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y} \qquad \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$$

The sets $L = \{(\parallel, \mathbf{0})\}$ and $R = \{(\parallel, \mathbf{0}), (\between, \mathbf{0})\}$ meet the constraints. Therefore, our theorem yields the soundness of the well known equations:

$$\mathbf{0} \parallel x = x, \quad x \parallel \mathbf{0} = x \quad \text{and} \quad x \between \mathbf{0} = x \ .$$

Note that the pair $(\between, \mathbf{0})$ cannot be added to L while preserving the constraints! Indeed, $\mathbf{0}$ is not a left unit for the left merge operator

Further Results and Future Work

1. We also have a rule format for zero elements to guarantee the validity of equations like

$$\text{deadlock}; x = \text{deadlock}.$$

   We have developed a variety of rule formats for other algebraic properties, such as (generalized) commutativity, distributivity, idempotent binary and unary operators.
2. Rule format guaranteeing determinism of transition relations.
3. Algorithms for generating axiomatizations of bisimilarity, possibly using the rule format for commutativity.
4. Meta SOS and PREG Axiomatizer tools.
5. Future work: Develop a theory of rule formats for languages with notions of names and binders. (First steps towards 'Nominal SOS'.)

Thanks and Shameless Self-Promotion

Take-home Message (reprise)

Using suitably tailored rule formats one can obtain semantic properties for classes of languages for free!

Read the papers available from

http://www.ru.is/faculty/luca/reports.html

(Search for 'Rule Format' or any of the keywords mentioned in this talk.)

Xie xie!
Any Questions?

Thanks and Shameless Self-Promotion

Take-home Message (reprise)

Using suitably tailored rule formats one can obtain semantic properties for classes of languages for free!

Read the papers available from

http://www.ru.is/faculty/luca/reports.html

(Search for 'Rule Format' or any of the keywords mentioned in this talk.)

Xie xie!
Any Questions?