A syntax-driven (and possibly solver-aided) approach to subtyping rate types

LUCAS DU, University of California, Davis, USA
DOOFENSCHMIRTZ DAVIES, The Couch, Catland, USA

Maybe a syntax-driven approach isn't so bad for subtyping rate type refinements? We present structural subtyping rules for rate types, a Kleene-like algebra for rewriting/simplifying rate types to a normal form, and a procedure for checking entailment between two rate types in normal form. The crux of this approach rests on ways in which we can abstract over problematic type constructors, namely: Concat ($\bullet$), Star (*), and Parallel ($\|$).

For Concat and Star, we introduce a rewriting rule when all components of the type (two for Concat, one for Star) are in "raw" form, i.e. of form $@n/t$, that eliminates the type constructor by introducing an equivalent conjunction. For Parallel, we treat it as an irreducible type until the moment we actually check subtyping/entailment; we then attempt to symbolically abstract over the Parallel type using a single symbolic "raw" rate type, generate some constraints needed for the entailment, and then solve those constraints to check entailment.

The hope is that (a) this algorithm is efficient and (b) that we can prove this syntax-driven abstract approach sound (and possibly complete) with respect to an underlying automata-based model like timed automata. For example, the Kleene-like algebraic rules that we rely on for rewriting/simplifying are not formalized in any way (they just "seem" right, which is actually kind of terrible); it seems quite important that we actually prove some kind of Kleene Theorem that gives these rules an actual basis.

1 Some motivation

1.1 Old problems

A big problem with our previous syntactic approach was that it lacked nice algebraic properties, such as compostionality. Another problem was that it was simply poorly designed, i.e. the initial formulation didn't take into account some basic things like (a) multiple rate type refinements on a single stream, (b) the fact that the stream types calculus allowed rate type refinements to be arbitrarily nested, and so were not all going to be "raw" rates of the form $@n/t$, where n and t are some natural numbers (or positive reals—this point is still not totally well-considered), or (c) the fact that rate refinements could have different window sizes (i.e. the older rule for uniform concatenation assumed the same window size for both streams and just added the events, see below).

S-UNIFORM-CONCAT-FACTOR

$$\frac{}{S_{@\mathsf{n}_1/\mathsf{t}} \bullet T_{@\mathsf{n}_2/\mathsf{t}} <: (S \bullet T)_{@(\mathsf{n}_1+\mathsf{n}_2)/\mathsf{t}}}$$

1.2 An aside: narrowing our focus

We will focus only on *uniform* (or *sliding*) rates for the moment, in order to simplify our work a bit. The hope is that generalizing ideas from the uniform (sliding) setting to the segmented (tumbling) setting won't be too much more of a lift.

1.3 Some new ideas

In response to these problems, we thought about ways to ground our type system in something more algebraic, i.e. Kleene algebras, regular languages, finite state automata, etc. This seems appealing,

October 10, 2025.
Authors' Contact Information: Lucas Du, lzdu@ucdavis.edu, University of California, Davis, USA; Doofenschmirtz Davies, The Couch, Catland, USA.

since most of our operations are covered (exactly) by the rules for Kleene algebras (see, for example: concatenation, star, and sum/choice), and thus finding some kind of model for rates in regular languages would give us all that nice theoretical goodness (somewhat, not exactly) for free.

This approach is still promising, and worth investigating much further, but there are a couple immediate issues (at least as far as we can tell, at the current moment):

- A very natural way to model rates is with *timed automata* (courtesy of Rajeev Alur and David Dill, 1994). But timed automata have some bad automata-theoretic properties, i.e. undecidability of inclusion for *nondeterministic* timed automata. And we (read: I) can't yet figure out how to model all the type constructs for our rates (i.e. Parallel and Concat) with a *deterministic* timed automata (which has better automata-theoretic properties).
- It's also possible to model rates as just a standard regular language/finite state automata using a notion of "ticks"—one "tick" per unit of time (represented by some letter like ✓), with events inside each unit of time (represented by some letter like a). So one possible trace could be: $\checkmark aaa\checkmark a\checkmark\checkmark\checkmark aa$. But a (possibly very naive) automaton construction for languages like this has exponential state complexity with respect to the number of "ticks" in a window: one upper bound on the state complexity is $(n+1)^m + 1$, for a uniform rate type of form $@n/m$.[1]

1.4 Back to syntax-driven typing rules (for now)

Given these potential efficiency problems with an automata-based approach, perhaps we should turn back temporarily to our old syntax-driven, typing rule style? I think there is a bit more juice there worth squeezing, particularly since it promises to be more efficient. The automata stuff is still important though, since I think it will give us a semantic basis to ground our syntactic typing rules.

2 A boolean-ish algebra for rate types

We define a core boolean-ish algebra for rate types, which we claim our structural subtyping rules and algebraic rewriting rules will reduce all rate types involved in subtyping entailments to. Here is the core rate type:

$$R ::= @n/t \mid @(n1/t1 \parallel n2/t2 \parallel \ldots) \mid R \vee R \mid R \wedge R \mid \top \mid \bot$$

Note that we also consider Parallel rates consisting only of "raw" rates to be irreducible here. Indeed, the idea is that we save all information about "raw" Parallel rates and propagate it until subtype checking time—the reasoning for this is that the subtype/entailment checking procedure for Parallel is not associative or compositional, so abstracting earlier in the process will lose possibly important information.

TODO: Add an example here to show how our abstraction procedure for Parallel is not associative or compositional.

Treating the first two symbols in our type ($@n/t$ and $@(n1/t1 \parallel n2/t2 \parallel \ldots)$) as base cases, this algebra is just a normal boolean algebra (although we don't have or need negation). For example, we can rewrite $(R_1 \vee R_2) \wedge R_3$ as $(R_1 \wedge R_3) \vee (R_2 \wedge R_3)$ (i.e. all the expected distributivity properties apply).

The expected rules for entailment (in boolean algebra) also apply, i.e for $R_1 \wedge R_2 \implies R_3$ is equivalent to $R_1 \implies R_3 \wedge R_2 \implies R_3$ (using the $\implies$ operator here to denote entailment).

[1] The main idea for this bound is that, at any moment in time, we need to hold m ticks, i.e. one window, in our memory, each tick of which could hold anywhere from 0 to n events. This is finite memory, so our language is regular, but encoding all possible configurations of this memory, along with a reject state, gives us a state complexity of $(n+1)^m + 1$. This would probably get worse when we try to model constructs like Concat or Parallel, or eventually do things like check inclusion.

3 Structural subtyping rules

The basic subtyping rules for "raw" (*uniform*) rate types, i.e. types of form $@n/t$, are as follows.

$$\frac{(t_2 < t_1 \wedge n_1 \leq n_2) \vee (t_1 \leq t_2 \wedge n_1 \leq n_2/(\lceil t_2/t_1 \rceil))}{@n_1/t_1 \leq @n_2/t_2} \text{ RAW-RATE-SUB}$$

4 Algebraic rewriting rules

4.1 Reduction to a normal form

5 Checking entailment

5.1 Checking entailment with parallel sums

6 Foundational theoretical concerns and next steps