Property Conveyances as a Programming Language

Shrutarshi Basu*
Harvard University
Cambridge, MA, USA
basus@seas.harvard.edu

Nate Foster
Cornell University
Ithaca, NY, USA
jnfoster@cs.cornell.edu

James Grimmelmann
Cornell Law School & Cornell Tech
New York, NY, USA
james.grimmelmann@cornell.edu

Abstract

Anglo-American law enables property owners to split up rights among multiple entities by breaking their ownership apart into *future interests* that may evolve over time. The *conveyances* that owners use to transfer and subdivide property rights follow rigid syntactic conventions and are governed by an intricate body of interlocking doctrines that determine their legal effect. These doctrines have been codified, but only in informal and potentially ambiguous ways.

This paper presents preliminary work in developing a formal model for expressing and analyzing property conveyances. We develop a domain-specific language capable of expressing a wide range of conveyances in a syntax approximating natural language. This language desugars into a core calculus for which we develop operational and denotational semantics capturing a variety of important properties of property law in practice. We evaluate an initial implementation of our languages and semantics on examples from a popular property law textbook.

CCS Concepts • **Applied computing** → **Law**; • **Software and its engineering** → **General programming languages**; • **Theory of computation** → *Denotational semantics*; *Operational semantics*; • **Human-centered computing** → Interactive systems and tools.

Keywords Domain-Specific Languages, Law, Semantics

ACM Reference Format:
Shrutarshi Basu, Nate Foster, and James Grimmelmann. 2019. Property Conveyances as a Programming Language. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '19), October 23–24, 2019, Athens, Greece.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3359591.3359734

*This work was started while the author was at Cornell University.

Onward! '19, October 23–24, 2019, Athens, Greece

ACM ISBN 978-1-4503-6995-4/19/10…$15.00
https://doi.org/10.1145/3359591.3359734

1 Introduction

Many legal issues depend on vague, open-ended standards. For example, nuisance law prohibits "unreasonable" interference with neighbors' use of land. What counts as "unreasonable" depends on an eight-factor balancing test, and many of the individual factors are themselves vague and open-ended, including "the social value that the law attaches to the primary purpose of the conduct;" and "the suitability of the particular use or enjoyment invaded to the character of the locality." [American Law Institute 1979] A lawyer who wants to know whether a client's cement plant will be considered a nuisance will have to research numerous previous cases, gather extensive facts about the plant and its neighbors, and rely on her accumulated intuitions about how courts tend to rule. Indeed, whether her client can build the plant may depend on the lawyer's skill in presenting an interpretation of the facts and the factors that favors the client.

Other legal issues depend on clearer and relatively unambiguous rules. For example, the tax code generally allows individuals to deduct the home mortgage interest they pay, but only on loans of up to $750,000. There is no room to argue over whether "$750,000" really should mean $75,000, $75,000,000, or a context-dependent amount. Tax law is full of unambiguous, mechanical calculations—this is why tax software is possible. These portions of tax law are effectively formalizable [Lawsky 2016].

In other areas of law, however, lawyers have been less quick to appreciate that the rules they work with have an underlying logical structure. Indeed, legal training, with its strong emphasis on careful reading and rhetorical skills, and its corresponding lack of emphasis on quantitative and formal methods, may actively discourage lawyers from thinking about legal rules in symbolic, rather than verbal, terms. Legal analysis tends to be inductive, rather than deductive, even where the law itself allows for truly deductive reasoning. The elegant structure of the rules themselves is obscured when they are analyzed with the open-ended techniques one might use to argue whether a land use is 'reasonable."

One such body of rules is the system of "estates in land" that govern the division of property rights over time. It is incredibly common for ownership of real estate (i.e. land and buildings) to be divided among multiple people and for the passage of time and the occurence of various events to change the division. For example, in a *lease*, both the landlord and the tenant have *interests* in the property, each of which comes with different rights. The tenant has the present right

to possess the property now; the landlord is not entitled to possession, but does have the right to receive periodic rent payments from the tenant. In the future, when the lease term ends, the tenant's right to possession will end, and the landlord will be entitled to reenter the property and retake possession. Or, the lease might provide that if a specified event occurs—for example, if an apartment tenant causes serious damage to the building—the tenant's interest might terminate immediately.

Law students learn the basics of the different types of estates in land in their property course, which is typically taught in the first year. They learn how to parse the legal documents (generically known as *conveyances*) by which one person can give her interest to another, or divide it up among several others, and how to give legally-meaningful descriptions of the resulting interests (such as "remainder in life estate subject to an executory limitation," meaning that one has the right to a piece for property for their lifetime, as long as they satisfy some additional condition). It takes several weeks to a month of class time and is frequently regarded as a tedious and pointless exercise in memorizing arcane rules [Grimmelmann 2017]. At the end of that time, students are hopefully able to read a conveyance such as "`O conveys to A for life, then to B for life, then to C for life.`" and report that it creates four interests:

1. A has a life estate.
2. B has a remainder in life estate.
3. C has a remainder in life estate.
4. O has a reversion in fee simple.

A "life estate" means that the interest is valid until the death of the possessor, while "fee simple" means that the interest is valid indefinitely (and passes to the possessor's legal heirs on their death). As previous legal scholars have noted, this example suggests the grammar of conveyances has a recursive structure: adding an additional clause (then to D for life) would create an additional interest (a remainder in life estate for D) [Edwards 2009; Merrill and Smith 2017].

Other conveyances have subtle consequences that are not obvious from a layperson's reading. Consider "`O conveys to A for life, then to B for life, but if C marries then to C`". This also creates four interests:

1. A has a life estate.
2. B has a remainder in life estate.
3. O has a reversion in fee simple.
4. C has an executory interest in fee simple.

One surprising twist here is that the event of C marrying would immediately terminate *both* A's and B's interests; another is that if both A and B were to die before C's marriage, then possession would temporarily revert to O.

This combination of a well-defined structure, and a subtle relationship between the syntax and semantics of conveyances, strongly suggests that tools from programming language theory may be useful in modeling conveyances. Some earlier work by the authors [Basu et al. 2017] described how the language of conveyances could be boiled down to an abstract syntax that allowed for some automated reasoning. But we did not provide semantics or derive interesting properties of our formalism. This paper provides a more formal treatment of estates in land through the lens of programming language theory. Our contributions are as follows:

- We develop a domain-specific language that is capable of expressing a wide range of realistic conveyances. It does not capture every turn of phrase a lawyer might use, but it covers the standard stock phrases that first-year law students become familiar with.
- The surface syntax for this language is ambiguous in ways that reflect well-known ambiguities in the natural language of actual conveyances. However, the ambiguities are limited in scope. We use a parser combinator library [Arnold and Mouratov 2008] to implement a parser for the language, one that covers a range of examples from a popular property law textbook.
- We develop a minimal core calculus that can express how the interests defined by a conveyance change in response to a sequence of events. Conveyances in the surface language desugar deterministically into a program in this core calculus.
- We develop an operational and a denotational semantics for programs in the core calculus and prove their equivalence. The operational semantics lead to a straightforward functional implementation. The denotational semantics allow us to derive useful properties of the language, and to give succinct proofs of several maxims used by lawyers to describe the functioning of the estates in land (e.g. "First in time, first in right.").

The rest of this paper is organized as follows:

1. Section 2 gives a number of examples of conveyances, explains the relevant terminology, and shows how they may be interpreted as programs in a formal language.
2. Section 3 describes in detail the surface syntax and the core calculus, and shows how the former desugars into the latter.
3. Section 4 discusses the operational and denotational semantics for the core calculus.
4. Section 5 discusses our current implementation and provides an initial evaluation. We evaluate the utility of the formalism by showing that common principles of property law can be formulated as theorems using it. We evaluate the implementation by testing it against a large number of examples from a popular property law coursebook.
5. Section 6 discusses related work from both computational and legal fields, and Section 7 concludes.

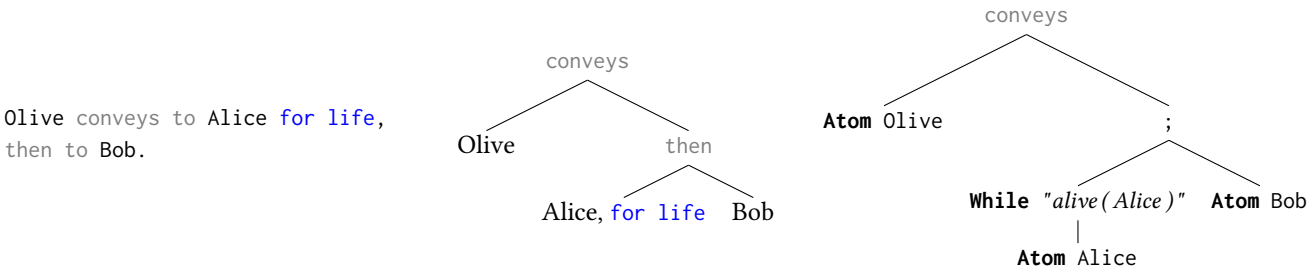


Figure 1. Parsing and desugaring a single conveyance.

2 Examples and Terminology

Property law deals with the rights of *owners* of *interests* in property. (We follow the terminology and the substantive law as described in [American Law Institute 1936], a historic and highly influential scholarly summary of United States property law.) The most important rights are typically the right to use a piece of property as one wishes and to exclude others from using it. These two rights are usually linked, and a person who has an interest that includes the rights to use and to exclude is said to be *entitled to possession* of the property, and the interest is said to be *possessory*. For real estate (land and things permanently attached to it, like buildings), an interest is called a *present estate* if it is possessory, and a *future interest* if it might become possessory sometime in the future (if at all). Interests are created (and the corresponding rights transferred) from one party (the *grantor*) to another (the *grantee*) in a variety of ways (wills, deeds, etc.), which we will collectively call *conveyances*.

We will follow the conventions of legal education and write conveyances in the simplified form that law students and legal textbooks commonly discuss them. For example, in the elementary conveyance "`Olive conveys to Alice for life, then to Bob`", Olive is the grantor and Alice and Bob are the grantees. The term "`conveys to`" indicates that Olive is transferring her interest. This conveyance has two *clauses*, separated by a comma for convenience. The first clause gives Alice an interest that will entitle her to possession starting immediately and continuing until Alice's death. Alice's interest is said to have a *natural duration*—Alice's lifetime, indicated by the term "`for life`". The second clause gives Bob an interest that will entitle him to possession starting at Alice's death and continuing forever (it will pass to his legal heirs upon his death). Alice's interest is presently possessory as it entitles her to possession now; Bob's is a future interest, since it will not entitle him to possession until Alice dies.

Our model of conveyances has two stages: first, we parse a conveyance written in (a very restricted subset of) English into an AST. Then we desugar the AST into a term in the core calculus. The parsing step identifies the clauses in the conveyance and their components; the desugaring step cleans them up into a standard representation.

Figure 1 shows parsing and desugaring for the example "`Olive conveys to Alice for life, then to Bob.`" The syntax `conveys to` indicates to the parser that this is a conveyance from owner Olive, and `then to` denotes a boundary between the two clauses. The first clause describes an interest owned by Alice, but guarded by a *condition*. The syntax `for life` indicates a *duration* that desugars into a condition attached to this clause. In this case, "for life" desugars to the condition "Alice is alive", referencing the owner. Our implementation provides desugarings for common durations and conditions, including "for life", "for the life of P" (where P is some person other than the grantee) and "for N years." At the end of the desugaring process, the AST generates the core language program on the right. For the rest of this paper we use gray to indicate connective keywords, blue to indicate durations, and green for other limiting events.

Conditions are language constructs that evaluate to true or false, depending on a history of events. For example, suppose the event "Alice dies" occurs. Once Alice has died, the condition "Alice is alive" no longer evaluates to true. Thus we can replace the subterm **`While`**`("Alice is alive", Alice)` with the empty subterm **`Bottom`**. This leaves the overall term as **`Bottom`**`; Atom Bob`, which simplifies to **`Atom`** `Bob`. This is typical of the operational semantics of terms: events can cause some conditions to become false, which in turn causes subterms to terminate. These terminated terms are replaced by **`Bottom`**, which yields possession to the following subterm in a sequence (the semicolon is the sequencing operator). If another event occurs—say "Olive dies"—no further changes take place, because **`Atom`** `p` always evaluates to **`Atom`** `p` under any possible event. Conditions and the operational semantics of terms are discussed in further detail in Section 4.

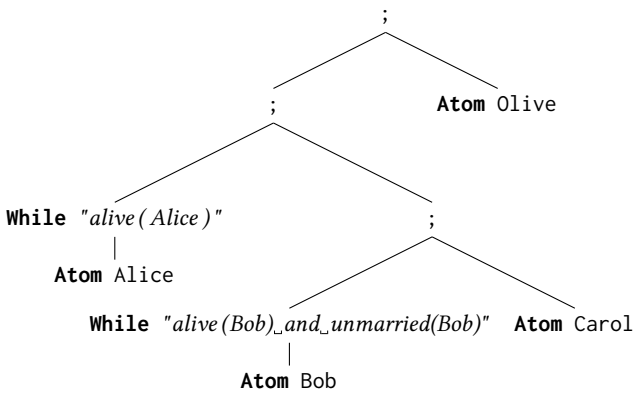
Here is a more complicated example:

```
1 Olive owns.
2 Olive conveys to Alice for life, then to Bob
    for life until Bob marries, then to Carol.
3 Alice dies.
4 Bob conveys to Dave for life.
5 Bob marries.
```

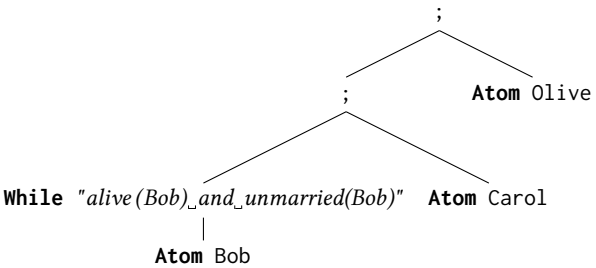
Let us trace the execution of this "program," step by step. It begins with an ownership statement. This is necessary for multi-part conveyances; otherwise, it is unclear whether the party named in each subsequent conveyance actually owns anything to convey. So after the first statement, the term is:

Atom Olive

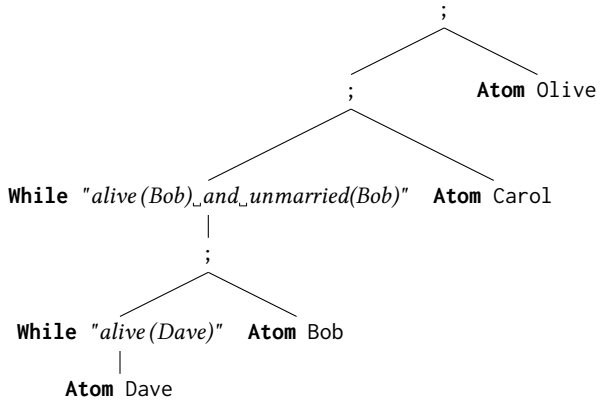
The second statement is a conveyance, but it is more complicated in two ways. First, it has three clauses, instead of two. The fact that the language of conveyances is recursive, in that additional clauses can be added indefinitely, is one of the key observations motivating our use of a context-free grammar for parsing conveyances. Second, the clause giving Bob an interest contains an *added limitation* ("until Bob marries"). This is an additional condition which could cut off Bob's interest "early" (i.e., before its natural duration). We model added limitations by adding an additional condition to the **While** node. Finally, the rule for a conveyance (for reasons to be explained shortly) is that the term created by the conveyance is added as a left sibling to the interest being conveyed. So after line 2, the term is



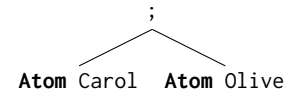
The third statement (`Alice dies`) is an event. It causes the condition *"alive(Alice)"* to become false, so the entire subterm **While** *"alive(Alice)"*—(**Atom** Alice) is replaced with **Bottom**. The overall term simplifies to:



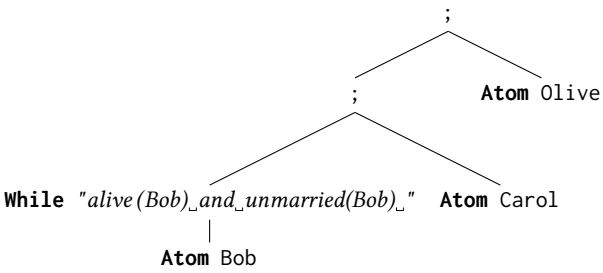
The fourth statement is another conveyance, but it raises two new complications. The first is that Bob does not convey all of his interest: he does not specify what happens to the property after Dave's death. This is where an important rule of property law, which we model, comes in. Bob is said to retain a *reversion*: a portion of his original interest that will become posesssory again if the interest he has conveyed to Dave ends. In effect, it is as though every conveyance ends with the implied words "`then to grantor`". This is why we add the new term as a left sibling to the interest being conveyed, rather than replacing it. The second complication is that Bob, who is conveying his interest, owns less than all of the property. So we add the new term created by Bob's conveyance as a left sibling to *Bob's interest* rather than to the overall term. The term is now:



The fifth statement is another event. It causes the condition *"alive(Bob) and unmarried(Bob)"* to fail, terminating all the interests that the condition guards. The complete term simplifies to:



Notice that Bob's death would also have yielded the same result. The failure of an outer condition causes all inner terms to be replaced with **Bottom** and be pruned away. Dave's death, on the other hand, would only have caused the subterm with Dave's interest to fail, so that the term would have become



This is *exactly the same* as the term before Bob's conveyance to Dave. This makes sense: Bob gave away part of his interest, but the part he gave away has terminated, so he is left in the same situation he was in before. In Section 5.1, we prove this is a general theorem about our model of property law. Note that the term **Atom** Carol ; **Atom** Olive will never simplify further, because no events are capable of terminating Carol's interest. Therefore, Olive's reversion is unreachable. Our implementation is capable of detecting and pruning unreachable interests (as lawyers do implicitly when talking about the state of title to a property), but we defer a full description of the required analysis to future work.

The next sections describe how we formalize the intuitions we have developed in this chapter, starting with a formalization of the syntactic structures we have discussed, and followed by operational semantics that allow us to perform the relevant evaluations and simplifications.

3 Concrete and Core Syntaxes

We have developed a surface syntax for expressing conveyances that that resembles natural English language and that desugars into terms in a core calculus. The surface syntax allows for specifying conveyances in a (very) restricted subset of English. These conveyances resemble the examples used in legal textbooks [Edwards 2009; Merrill and Smith 2017], and most of our examples are legally sufficient to do what they claim to. (Having said that, please don't try this at home—this is an academic paper, not legal advice!)

Figure 2b shows the concrete syntax of our surface language, which is designed to be familiar to legal practitioners. A "program" consists of an initial ownership declaration followed by zero or more statements. A statement can represent either the occurrence of an event e, or a conveyance. Thus events and conveyances can be interleaved, as is the case in the real world. A conveyance is a pair of a *grantor* p and a combination of clauses qs. Each clause has a grantee (to p) and optionally, a precondition (if c), a duration (for d) and a limitation (while c). The clauses can be singletons, or linked in various combinations, representing the common syntactic forms (as seen in Section 2). The syntax supports sequential composition ("q then qs"), or guarded composition, where the fulfillment of a condition cuts short all previous interests ("but if c then", i.e., an executory interest).

The grammar of conditions includes a few primitives that cover conditions common in practice (e.g., "p is married"), a primitive that is true when the corresponding event occurs, and operators for combining simpler conditions (including the standard logical operators). Our implementation is a little more flexible than the syntax shown in Figure 2. For example, a condition that is true as long as a person p is in school may be phrased as "while p is in school", as shown in Figure 2, but also as "until p graduates", or "so long as p is in school".

One of our core insights is that although many conveyances can appear complex in the surface syntax, they can usually be translated into programs in a simpler core language with a small collection of basic forms, as shown in Figure 2c. The treatment of programs, statements, conditions, and events is essentially unchanged. However, the numerous special cases associated with clauses, combinations, and durations have been simplified as terms in the core language.

Terms elide most syntactic characteristics of conveyances in favor of preserving the core semantic differences. The term `Atom` γ encodes what property law practitioners would recognize as a distinct interest: a triple of unique identifier ($n \in \mathbb{N}$), a grantor ($g \in \mathbb{P}$), and an owner (or grantee) ($o \in \mathbb{P}$).

Naturals	$\mathbb{N} \ni n$	::=	$0 \mid 1 \mid 2 \ldots$
Persons	$\mathbb{P} \ni p, g$	::=	$O, G, P, A, B \ldots$
Strings	x, y, z	::=	Strings
Events	$E \ni e$	::=	p dies $\mid$ n years pass $\mid$ x occurs $\mid \ldots$
Histories	$E^* \ni \bar{e}$	::=	$[e_0, \ldots, e_n]$

(a) Common Types.

Durations	d	::=	life $\mid$ the life of p $\mid$ n years
Conditions	c	::=	p is married $\mid$ p is alive $\mid$ e occurs
		$\mid$	$\neg c \mid c_1 \wedge c_2 \mid c_1 \vee c_2$
Clauses	q	::=	[if c] to p [for d][while c]
Combinations	qs	::=	$q \mid q$, then qs
		$\mid$	qs, but if c then qs
Statement	s^{nat}	::=	$(p,\ qs) \mid e$ occurs
Program	π^{nat}	::=	[Owns p; $s_0^{\text{nat}}; \ldots s_k^{\text{nat}}$]

(b) Natural Language Syntax.

Interest	$\Gamma \ni \gamma$	::=	To $(n \in \mathbb{N}, g, o \in \mathbb{P})$	Transfer
Terms	$t \ni T$	::=	Atom γ	One Interest
		$\mid$	Seq (t_1, t_2)	Sequencing
		$\mid$	While (c, t)	Termination
		$\mid$	If (c, t)	Precondition
		$\mid$	Bottom	Reversion
Statement	$S \ni s$	::=	Conveys $(\gamma,\ t) \mid e$	
Program	π	::=	[Owns p; $s_0 \ldots s_n$]	

(c) Core Language Syntax.

Figure 2. Syntax for Expressing Conveyances.

The additional structure is necessary because some property doctrines depend on whether an interest is owned by its grantor and on the identity of specific interests over time. Placing this information in the interest itself, rather than reconstructing it from context in a term, allows for simpler semantics. The unique identifiers allow for a program-level property: an interest can only be conveyed once, i.e., γ can appear on the right-hand-side of only one Conveys (γ,t) statement in a program. The term `Seq` (t_1, t_2) represents the linear sequence of the subterms t_1 and t_2: i.e., first the interests in t_1 are possessory, and then those in t_2 are. For concision, we often write $(t_1\ ;\ t_2)$. The term `While`(c, t) represents the termination of the subterm t on the failure of the condition c: if t is possessory and c becomes false, then the entire term ends. Conversely, `If`(c, t) evaluates the subterm t

$$
\begin{array}{rcl}
\gamma(\cdot) & : & \text{Person } (\mathbb{P}) \to \text{Interest } (\Gamma) \\
\nu(\cdot) & : & \text{Person } (\mathbb{P}) \to \text{Interest } (\Gamma) \\
\mathcal{D}[\![\cdot]\!] & : & \text{Duration} \to \text{Condition} \\
\mathcal{T}_n[\![\cdot]\!] & : & \text{Natural} \to \text{Core} \\
\mathcal{T}_q[\![\cdot]\!] & : & \text{Person} \to \text{Combination} \to \text{Term} \\
\mathcal{T}_c[\![\cdot]\!] & : & \text{Clause} \to \text{Term}
\end{array}
$$

$$
\mathcal{T}_n[\![\text{Owns } o; \bar{s}]\!] = \text{Atom } \nu(o); \mathcal{T}_n[\![\bar{s}]\!]
$$

$$
\begin{array}{rcl}
\mathcal{T}_n[\![s_1; s_2; \ldots; s_n]\!] & = & \mathcal{T}_s[\![s_1]\!]; \mathcal{T}_s[\![s_1]\!]; \ldots; \mathcal{T}_s[\![s_n]\!] \\
\mathcal{T}_n[\![e]\!] & = & e \\
\mathcal{T}_n[\![\text{Conveys } (g, qs)]\!] & = & (\gamma(g), \mathcal{T}_q[\![qs]\!]_g)
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{T}_q[\![q]\!]_g & = & \mathcal{T}_c[\![q]\!] \\
\mathcal{T}_q[\![q, \text{ then } qs]\!]_g & = & \mathcal{T}_c[\![q]\!];\ \mathcal{T}_q[\![qs]\!]_g \\
\mathcal{T}_q[\![qs_1, \text{ but if } c \text{ then } qs_2]\!]_g & = & \text{While } (\text{not } c, (\mathcal{T}_q[\![qs_1]\!]_g; \text{Atom } \nu(g)));\ \mathcal{T}_q[\![qs_2]\!]_g
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{T}_c[\![\_, \text{to } p, \text{for } d, \text{while } c]\!] & = & \text{While } (c \vee \neg\mathcal{D}[\![d]\!], \text{Atom } \nu(p)) \\
\mathcal{T}_c[\![\_, \text{to } p, \_, \text{while } c]\!] & = & \text{While } (c, \text{Atom } \nu(p)) \\
\mathcal{T}_c[\![\_, \text{to } p, \text{for } d, \_]\!] & = & \text{While } (\neg\mathcal{D}[\![d]\!], \text{Atom } \nu(p)) \\
\mathcal{T}_c[\![\_, \text{to } p, \_, \_]\!] & = & \text{Atom } \nu(p) \\
\mathcal{T}_c[\![\text{if } c, p, d, c']\!] & = & \text{If } (c,\ \mathcal{T}_c[\![(\_, p, d, c')]\!])
\end{array}
$$

Figure 3. Translating from Natural to Core Language.

only if the condition c is true; if not, the entire term ends immediately. `Bottom` represents the case when all interests in a subterm have terminated. It is used only internally within the semantics while processing the effects of events. As we shall see in Section 4, the semantics prune away constructs that reduce to `Bottom`. Finally, a conveyance is a pair of the interest γ owned by the grantor and the term t representing the interests created by the conveyance.

Figure 3 shows the translation from natural language to the core programs. There are a number of important subtleties. First, conveyances of the form "qs_1 but if c then qs_2" make use of the `While`(c, t) construct, (not the `If`(c, t) construct, as would be intuitive) enclosing all preceding terms as the body term. This is to correctly implement the legal interpretation that condition c becoming true terminates all preceding interests (as shown in the example in Figure 1). This translation also inserts a transfer back to the grantor g after the translation of qs_1. Intuitively, this accounts for the case when all the interests represented by qs_1 might have been terminated (due to their own durations and limitations), but the condition c has not yet been fulfilled.

However, a different translation applies for syntax of the form "if c to p" as in the final case (i.e., when the condition c is part of the clause, rather than appearing before it). In this case, the translation to the core `If`(c, t) construct implements a requirement that the condition c hold *at the time* that the wrapped term t is to be evaluated (i.e., when all preceding interests have been terminated).

Translating the remaining forms of clauses according to $\mathcal{T}_c[\![\cdot]\!]$ is more straightforward. All clauses require a grantee (to p), but durations (for d) and conditions (while c) are optional. We elide the duration translation function $\mathcal{D}[\![\cdot]\!]$ for concision. There are a small number of possible durations (as shown in Figure 2); $\mathcal{D}[\![\cdot]\!]$ simply hard-codes their corresponding conditions. When both a duration and condition are present, their conditions are conjoined. Conditions are explained in greater detail in Section 4.1.

The translation also makes use of two book-keeping functions. The function $\nu(\gamma)$ generates a fresh interest γ' in the core language such that γ and γ' share the same owner. Fresh interests are necessary because the same person could hold two or more distinct interests. The function $\gamma(g)$ picks out a specific interest conveyed by the person g and is necessary for the same reason: one person could hold two or more distinct interests. Currently, $\gamma(g)$ returns the most recent interest owned by g known to the system. It could be extended (along with the Conveys statement) to allow particular interests to be chosen without altering the rest of the formalism.

The translation from concrete to core syntax is straightforward, but not trivial. It requires understanding the intricacies of the language used in conveyances, and depends on domain-specific knowledge, often producing results that are different from a non-expert reading of the legal text. As we shall see in the next section, the design of the core syntax supports an operational semantics that closely matches the legal interpretation of the corresponding terms.

$$P \;:\; \Pi \to \Gamma$$

$$P(\text{Owns } o; \bar{s}) \;=\; \begin{cases} o & \text{if } \gamma = \bot \\ \gamma & \text{otherwise} \end{cases} \quad \text{where } \gamma = E(\Delta_{\bar{s}}(\text{Atom } o))$$

(a) Program evaluation function P.

$$E \;\in\; T \to \Gamma \cup \{\bot\}$$

$$\begin{aligned} E(\,\text{Bottom}\,) &= \bot \\ E(\,\text{Atom } \gamma\,) &= \gamma \\ E(\,t_1\,;\,t_2\,) &= E(t_1) \\ E(\,\text{If}(c,t)\,) &= E(t) \\ E(\,\text{While}(c,t)\,) &= E(t) \end{aligned}$$

(b) Observation function E.

$$\Delta_{\bar{s}} \;:\; T \to T$$

$$\begin{aligned} \Delta_{\epsilon}(t) &= \delta(t) \\ \Delta_{\bar{s}s}(t) &= \delta(\langle t' \rangle_{s'}) \\ &\quad \text{where } t' = \Delta_{\bar{s}}(t) \\ &\quad \text{and } s' = \langle s \rangle_{\bar{s}} \end{aligned}$$

(c) Transition function Δ.

Figure 4. Operational semantics of conveyance programs.

4 Operational and Relational Semantics

The previous section showed how to construct terms in the core language that represent the legal understanding of what the equivalent natural language conveyances should do. In this section, we develop two forms of semantics for the core language. These semantics define how these terms represent the changing of interests in response to sequences of events.

In Section 4.2, we develop an operational semantics, based loosely on the notion of derivatives of regular expressions [Antimirov 1996; Brzozowski 1964]. This operational semantics translates directly to a concrete implementation discussed in Section 5. Then, in Section 4.3 we develop a denotational semantics that relates sequences of events and the corresponding possessory interests represented by a term. This semantics allows us to prove that our model obeys a number of important and widely-accepted concepts in property law (as shown in Section 5.1). Finally, in Section 4.4 we show that the two semantics are equivalent. This guarantees that an implementation of the core language that obeys the operational semantics will conform to the expectations of legal practitioners.

4.1 Conditions

Before we dive into the semantics, a discussion of conditions is warranted. In reality, conditions can be arbitrarily complex, but there are a number of conditions that are common in the practice of property law, e.g., marriages, divorces, births and deaths. Fully characterizing the space of possible conditions is outside the scope of this work. Conceptually, we model conditions as functions from sequences of events to a boolean: $E^* \to \{\texttt{true}, \texttt{false}\}$. As demonstrated in the previous sections, evaluating a term often requires checking whether a condition is true or false. We write $\bar{e} \models c$ when $c(\bar{e}) = \texttt{true}$. When $\bar{e} = \epsilon$ we write simply $\models c$.

We will also be concerned with how conditions evolve as events occur. Thus, we also need some notion of "stepping" them forward on single events. We require each condition c to have an associated stepping function $\langle c \rangle_e$ such that $\bar{e} \models \langle c \rangle_e$ iff $e\bar{e} \models c$. The stepping function $\langle \cdot \rangle$ lifts from events to sequences of events in the obvious way.

In many cases, conditions can be defined as automata over a sequence of events. In that case, conditions could be expressed with a language similar to regular expressions.

$$\begin{aligned} c \;::=\; & \text{Now } e \mid \text{Occurred } e \\ \mid\; & \neg c \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \end{aligned}$$

$$\models \;\in\; C \times E^* \to \{\texttt{true}, \texttt{false}\}$$

$$\begin{aligned} \bar{e} &\models \text{Now } e \;\text{ iff }\; \bar{e} = \bar{e}'e \\ \bar{e} &\models \text{Occurred } e \;\text{ iff }\; e \in \bar{e} \\ \bar{e} &\models \neg c \;\text{ iff }\; \bar{e} \not\models c \\ \bar{e} &\models c_1 \wedge c_2 \;\text{ iff }\; \bar{e} \models c_1 \text{ and } \bar{e} \models c_2 \\ \bar{e} &\models c_1 \vee c_2 \;\text{ iff }\; \bar{e} \models c_1 \text{ or } \bar{e} \models c_2 \end{aligned}$$

Figure 5. Defining simple conditions over events.

The stepping function would be similar to the notion of derivatives of regular expressions. In this paper we treat conditions abstractly, and leave exploring this direction to future work. Specifically, our current implementation provides hand-coded implementations of some common conditions and the related stepping functions (eg., births, deaths, marriages and divorces). The implementation allows for defining new events (as opaque strings) and new conditions using boolean operators in terms of those events, as in Figure 5. This provides an "escape hatch" for complicated conditions, and lets us focus on the semantics of conveyances, rather than dealing with the intricacies of real-world conditions.

4.2 Operational Semantics

These semantics are based on the following principles:

1. At any given time, exactly one interest is *possessory* (i.e., the owner of that has interest the right to use the property at that time). A term is sufficient to determine the current possessory interest: it is the leftmost non-terminated interest in the term.
2. Terms change when particular conditions become true (or false) in response to events, and when interests are conveyed—i.e., when they are syntactically replaced by other terms.
3. Thus, we can think of *applying* a term to a statement to produce another term. A term t' produced in this way is the *derivative* of the original term t with respect to a statement s. Here, t' denotes the possible future interests allowed by t after statement s has occurred.

$$\langle\cdot\rangle_s \;:\; T \to T$$

$$\begin{aligned}
\langle \text{Atom } \gamma\rangle_e &= \text{Atom } \gamma \\
\langle \text{Atom } \gamma\rangle_{(\gamma',t)} &= \text{Atom } \gamma && \text{if } \gamma \neq \gamma' \\
\langle \text{Atom } \gamma\rangle_{(\gamma,t)} &= t\,;\,\text{Atom } \gamma' && \text{where } \gamma' = \nu(\gamma)
\end{aligned}$$

$$\begin{aligned}
\langle \text{Bottom}\rangle_s &= \text{Bottom} \\
\langle \text{If}(c,t)\rangle_s &= \text{If}(\langle c\rangle_s, \langle t\rangle_s) \\
\langle \text{While}(c,t)\rangle_s &= \text{While}(\langle c\rangle_s, \langle t\rangle_s) \\
\langle t_1\,;\,t_2\rangle_s &= \langle t_1\rangle_s\,;\,\langle t_2\rangle_s
\end{aligned}$$

(a) Stepping a term by one statement

$$\begin{aligned}
\langle t\rangle_\epsilon &= t \\
\langle t\rangle_{\bar{s}s} &= \langle\langle t\rangle_{\bar{s}}\rangle_{\langle s\rangle_{\bar{s}}}
\end{aligned}$$

(b) Stepping a term by a sequence of statements.

$$\begin{aligned}
\langle\cdot\rangle_s &: S \to S \\
\langle e\rangle_s &= e \\
\langle(\gamma,t)\rangle_s &= (\gamma, \langle t\rangle_s)
\end{aligned}$$

(c) Stepping one statement.

$$\begin{aligned}
\langle\cdot\rangle_s &: S^* \to S^* \\
\langle\epsilon\rangle_s &= \epsilon \\
\langle\bar{s}s'\rangle_s &= \langle\bar{s}\rangle_s\langle s'\rangle_s
\end{aligned}$$

(d) Stepping a sequence of statements by one statement.

$$\begin{aligned}
\langle\cdot\rangle_{\bar{s}} &: S \to S \\
\langle s'\rangle_\epsilon &= s' \\
\langle s'\rangle_{\bar{s}s} &= \langle\langle s'\rangle_{\bar{s}}\rangle_{\langle s\rangle_{\bar{s}}}
\end{aligned}$$

(e) Stepping one statement by a sequence of statements.

Figure 6. Stepping functions $\langle\cdot\rangle$ for terms and statements.

The semantics of an entire program are defined by a program evaluation function P shown in Figure 4a. P starts with an initial ownership as defined by the initial `Owns` statement, applies transition function Δ to evaluate the effects of subsequent events and conveyances, and then observes the resulting possessory interest with observation function E(defined in Figures 4b and 4c respectively.)

This formulation is based loosely on the notion of Antimirov partial derivatives for regular expressions [Antimirov 1996]. $\Delta_a(r)$ is the *derivative* of a regular expression r with respect to the character a. It is the language that results from removing the character a from from all those words that start with a in the language r. $E(r) \in \{0, 1\}$ *observes* if the empty string is accepted by the regular expression r, i.e., if r is already satisfied. Together they define how a regular expression changes as a stream of characters are observed.

In our setting, Δ describes how a term changes with respect to a list of statements (events or conveyances) by producing a new term, while E gives the current possessory interest. E is straightforward. Recall from Section 3 that a conveyance is a pair of a grantor and a term in the core language. If the term has terminated, E returns the grantor, otherwise it returns the current possessory interest by recursively traversing the term.

The transition function Δ alternately steps a term t by one statement s using $\langle\cdot\rangle$ and simplifies the result with δ to apply any changes required by the state of the conditions in the term. Δ is complicated by the fact that a statement can either be a single event, or another conveyance, which might insert a new subterm. The Δ function is applied recursively to all but the final statement, producing a penultimate term t'. This term is then stepped, not with the final statement s, but with that final s itself stepped by all previous statements $\bar{s}$. This ensures that if s is a conveyance with a term, conditions in that term are stepped according to all previous statements.

The simplification function $\delta : T \to T$ defined in Figure 7 is straightforward: it removes the leftmost subterm when required by a failing condition.

$$\begin{aligned}
\delta &: T \to T \\
\delta(\text{Bottom}) &= \text{Bottom} \\
\delta(\text{Atom } \gamma) &= \text{Atom } \gamma \\
\delta(\ t_1\,;\,t_2\ ) &= \text{if } \delta(t_1) = \text{Bottom} \text{ then } \delta(t_2) \\
&\qquad \text{else } \delta(t_1)\,;\,t_2 \\
\delta(\ \text{If}(c,t)\ ) &= \text{if } \models c \text{ then } \delta(t) \text{ else Bottom} \\
\delta(\ \text{While}(c,t)\ ) &= \text{if } \models c \text{ then While}(c, \delta(t)) \\
&\qquad \text{else Bottom}
\end{aligned}$$

Figure 7. Simplification function δ.

The stepping function $\langle\cdot\rangle$ is defined in Figure 6. First, $\langle\cdot\rangle_s : T \to T$ in 6a steps a term. For terms that are not Atom, $\langle\cdot\rangle_s$ steps all the conditions and subterms in the term by the given statement. The Atom cases are more complex. If the statement is an event, the term is unchanged. If the statement is a conveyance, but for an interest other than the one wrapped by the Atom, the term is also unchanged. If the statement is a conveyance for the wrapped interest γ, then Atom γ is replaced by the term t in the conveyance, followed by a term, Atom γ', representing a *reversion* to the owner of the original interest γ. This γ' is a fresh, unique interest, different from both the conveyed interest γ and any other interest γ'' created elsewhere in the program. To create it, we use the function ν described in Section 3.

Figure 6b shows how to step a term by a sequence of statements. This is analogous to Figure 6e (described below) and required for the equivalence proof in Section 4.4.

$$
\begin{array}{rcl}
\mathcal{P}[\![\cdot]\!] & : & \Pi \to \Gamma \cup \{\bot\} \\
C[\![\cdot]\!] & : & T \to (S^* \times S^*) \to \{\texttt{true}, \texttt{false}\} \\
[\![\cdot]\!] & : & T \to (S^* \times S^*) \to \Gamma \cup \{\bot\} \\
[\![\cdot]\!]_{\not\perp} & : & T \to S^* \to \mathcal{P}(S^*) \\
B & : & \mathcal{P}(S^*) \to \mathcal{P}(S^*)
\end{array}
$$

$$
\mathcal{P}[\![\text{Owns } o; \bar{s}]\!] = [\![\text{Atom } o]\!](\epsilon, \bar{s})
$$

$$
C[\![c]\!](\bar{s}_0, \bar{s}_1) = \text{if } \forall \bar{s}_1' \le s_1 : \bar{s}_0 \bar{s}_1' \models c \text{ then } \texttt{true} \text{ else } \texttt{false}
$$

$$
\begin{array}{rcl}
[\![\text{Bottom}]\!](\bar{s}_0, \bar{s}_1) & = & \bot \\
[\![\text{Atom } \gamma]\!](\bar{s}_0, \bar{s}_1) & = & \text{if } \exists \min \bar{s}_0' : \bar{s}_0 = \bar{s}_0'(\gamma, t')\bar{s}_0'' \text{ then } [\![t'\,;\, \text{Atom } \gamma]\!](\bar{s}_0'\bar{s}_0'', \bar{s}_1) \\
 & & \text{else if } \exists \min \bar{s}_1' : \bar{s}_1 = \bar{s}_1'(\gamma, t')\bar{s}_1'' \text{ then } [\![t'\,;\, \text{Atom } \gamma]\!](\bar{s}_0\bar{s}_1', \bar{s}_1'') \\
 & & \text{else } \gamma \\
[\![\text{If}(c, t)]\!](\bar{s}_0, \bar{s}_1) & = & \text{if } \bar{s}_0 \models c \text{ then } [\![t]\!](\bar{s}_0, \bar{s}_1) \text{ else } \bot \\
[\![\text{While}(c, t)]\!](\bar{s}_0, \bar{s}_1) & = & \text{if } C[\![c]\!](\bar{s}_0, \bar{s}_1) \text{ then } [\![t]\!](\bar{s}_0, \bar{s}_1) \text{ else } \bot \\
[\![t_1\,;\, t_2]\!](\bar{s}_0, \bar{s}_1) & = & \text{if } \exists \bar{s}_1' : \bar{s}_1 = \bar{s}_1'\bar{s}_1'' \wedge \bar{s}_1' \in B(T([\![t_1]\!]_{\not\perp}(s_0))) \text{ then } [\![t_2]\!](\bar{s}_0\bar{s}_1', \bar{s}_1'') \text{ else } [\![t_1]\!](\bar{s}_0, \bar{s}_1)
\end{array}
$$

$$
\begin{array}{rcl}
[\![t]\!]_{\not\perp}(\bar{s}_0) & = & \{\bar{s}_1 \mid [\![t]\!](\bar{s}_0, \bar{s}_1) \neq \bot\} \\
T(L) & = & \{x \mid \forall y \ge x : y \notin L\} \\
B(L) & = & \{x \mid \forall y < x : y \in L\}
\end{array}
$$

Figure 8. Relational Semantics

Function $\langle\cdot\rangle_s : S \to S$ defined in Figure 6c steps a *statement* by stepping any terms that are part of a conveyance. This gives the correct semantics with respect to events that have happened when the conveyance is executed.

There are two ways to lift the stepping function to sequences of statements. Stepping a sequence of statements by one statement is straightforward and is described in Figure 6d. However, stepping a single statement s' by sequences of statements $\bar{s}s$ is more complicated: before s' can be stepped by s, the statement s must itself be stepped by all preceding statements in $\bar{s}$. This second lifting is formalized in the function $\langle\cdot\rangle_{\bar{s}} : S \to S$ in Figure 6e and depends on the preceding lifted function $\langle\cdot\rangle_s : S^* \to S^*$ for the base case. This is required because programs will be composed of sequences of statements, and evaluating a program requires stepping an initial ownership term by the statements in the program.

4.3 Relational Semantics

We can also define a relational semantics $[\![\cdot]\!]$ that regards each term in the core language as specifying a function from a history (a past sequence of events) and a future (also a sequence of events) to the interest that will be possessory after the future (or $\bot$ if no interest decribed by the term will be possessory). This relational formulation uses a denotation function $C[\![\cdot]\!]$ for conditions, which describes (given a history) the futures on which a condition always evaluates to `true`. The semantics also makes use of a boundary function $B(\cdot)$ which informally describes the futures that cause a term to irrevocably fail so that it can no longer result in any possible possessory interests. The most natural way to build up the boundary function is in three steps. First, the denotation function $[\![t]\!]_{\not\perp}(\bar{s}_0)$ returns the set of futures s_1 for which $[\![t]\!](\bar{s}_0), \bar{s}_1)$ has a value other than $\bot$. Then, treating this set as a language L, the set $T(L)$ of *terminated* strings is the set of strings whose extensions (including themselves) are not in the language. We then take the boundary $B(T(L))$ of the set of terminated strings: the strings that are in $T(L)$ but none of whose prefixes are.

4.4 Relating the Two Semantics

Next we prove a theorem relating the two semantics:

Theorem 4.1. $E(\Delta_{\langle\bar{s}_1\rangle_{\bar{s}_0}}(\langle t\rangle_{\bar{s}_0})) = [\![t]\!](\bar{s}_0, \bar{s}_1)$.

Note how each semantics keeps track of the sequence of past ($\bar{s}_0$) and future ($\bar{s}_1$) statements. The proof, which goes by induction on t, $|\bar{s}_0|$, $|\bar{s}_1|$, is given in the appendix.

5 Implementation and Evaluation

We have developed an implementation of the parser and semantics. Our implementation accepts conveyances written in the simplified natural language syntax described in Section 3, and performs the translation to the core syntax. The system then executes each statement in the program, one at a time, producing each intermediate term. Additionally, we have developed a user interface atop the implementation that adds some key features. The user interface is discussed in more detail in Section 5.3.

The core system is implemented in about 2600 lines of OCaml. The web-based user interface is implemented in about 300 lines of HTML, CSS, and JavaScript. It is accessible online at `https://conveyanc.es`.

We evaluate our work in two ways. First, we phrase a number of common legal principles in terms of our syntax and semantics, and show that they hold as theorems in our formalism. Second we have translated a number of examples from a popular property law textbook and show that our implementation produces the expected answers. Finally, we show some output from the user interface, highlighting some aspects that extend beyond our formalism and point to the direction of possible future work.

5.1 Legal Theorems

Our treatment of conveyances enables us to provide formal models and "proofs" of commonly recited legal propositions. We use the term "proof" lightly—no formal model or mathematical proof is capable of predicting or constraining what courts do in all cases, and we have not yet written down detailed proofs using the formal semantics for each of these propositions. Instead, the point of these examples is to show that our formalization can be used to correctly capture shared understandings of property lawyers and scholars.

A fee simple is perpetual and unconditional. Unless she conveys it away, the current owner of a fee simple will be the owner forever, regardless of how they acquired it and no matter what else happens. In our model, this is equivalent to the statement that $[\![\text{Atom } \gamma]\!](\bar{s}_0, \bar{s}_1) = \gamma$ for all $\bar{s}_0 \in S^*$ and $\bar{s}_1 \in S^*_\gamma$. The notation S_γ refers to $E \cup \{(\gamma', t) | \gamma' \neq \gamma\}$, i.e., all statements except a conveyance of γ. The proof is trivial by inspection of the Atom γ case in the definition of $[\![\cdot]\!]$.

Ownership is always unambiguous. No matter what has happened or will happen, someone will always be entitled to possession of the property. It is never the case that more than one party is entitled to possession, or that no one is. In our model, this requires that $\forall \pi \in \Pi. \mathcal{P}[\![\pi]\!]$ exists, is single-valued, and is not equal to $\perp$. The proof is straightforward: $\mathcal{P}[\![\text{Owns o}; \bar{s}]\!] = [\![\text{Atom } o]\!](\epsilon, \bar{s})$. Existence and single-valuedness follow from the equivalence of the operational and denotational semantics. Inspection of the definition of $[\![\text{Atom } o]\!]$ shows that it can never have the value $\perp$.

Nemo dat quod non habet. ("no man can give what he does not have"). No person can execute a conveyance that will eliminate someone else's right to possession, now or in the future. In our model, this is equivalent to the statement that if $[\![t]\!](\bar{s}_0, \bar{s}_1) = \gamma$ and $\gamma \neq \gamma'$, where $\bar{s}_0 \in S^*$ and $\bar{s}_1 \in E^*$, then $[\![t]\!](\bar{s}_0, (\gamma', t')\bar{s}_1) = \gamma$. The proof relies on an important property of the semantics of terms: $\bar{s}_1 \in ([\![t]\!]_\perp(\bar{s}_0)$ if and only if $(\gamma, t')\bar{s}_1 \in ([\![t]\!]_\perp(\bar{s}_0))$. This property states, informally, that a conveyance cannot change *whether* some interest will be possessory according to a term, only *which* interest will be possessory. The main non-trivial case is Atom γ, and the key observation is that $[\![\text{Atom } \gamma]\!]_\perp = S^*$. The next step is to recognize that $T(\cdot)$ and $B(\cdot)$ are also invariant under prefixing by a conveyance, which follows immediately from their definitions. From here, the main property should follow by an induction on t. The cases for Bottom, If(c, t), and While(c, t) are trivial. The case for Atom γ splits into simple cases depending on whether $\gamma = \gamma'$, and the case for $t_1 \,;\, t_2$ follows from the invariance of $B(\cdot)$.

First in time, first in right. If a party attempts to convey the same interest twice in succession, the first conveyance takes priority. In our model, this can be captured by the statement that if $[\![t]\!](\bar{s}_0, (\gamma, t')\bar{s}_1) = \gamma'$ for $\bar{s}_0, \bar{s}_1 \in S^*$, and $\gamma \neq \gamma'$, then $[\![t]\!](\bar{s}_0, (\gamma, t')(\gamma, t'')\bar{s}''_1) = \gamma'$. The proof, like the principle, is a special case of *nemo dat*.

Conservation of estates. If an owner of a present or future interest conveys away less than all of her interest, she retains a reversion that is entitled to possession if none of the interests she has created is. In our model, this is equivalent to the statement that if $[\![t]\!](\bar{s}_0, \bar{s}_1\bar{s}_2) = \gamma$ and $[\![t']\!](\bar{s}_0, \bar{s}_1) = \perp$, for $\bar{s}_0, \bar{s}_1, \bar{s}_2 \in S^*$, then $[\![t]\!](\bar{s}_0, \bar{s}_1(\gamma, t')\bar{s}_2) = \gamma$. The proof is immediate from the definition of $[\![\cdot]\!]$.

5.2 Textbook Examples

The textbook *Estates in Land and Future Interests: A Step-by-Step Guide* [Edwards 2009] is often used to teach property law as part as the first-year law curriculum. There are six chapters in first half of the textbook that fall within the scope of our work. For our evaluation, we took examples in the text and exercises in each chapter, and made any minor changes necessary to write them as inputs in our concrete syntax. We compared the output of our implementation, in terms of interests after evaluation, against the provided answers in the textbook. We consider our implementation successful if we produce the same set of interests as in the textbook.

Figure 9 summarizes the results of the evaluation. For each chaper, we detail the total number of test cases, the number of successes and failures. As we see, the implementation is successful in most of these cases. However, for some of them, we need to make minor changes to the syntax from the textbook to work with our implementation (the Smoothing column). In most cases, this involved removing minor punctuation like commas and semicolons. In some other cases, we need to add an event into the program before evaluating the conveyances (the Positive column). In general, this is to allow for what would be "common sense" reasoning, or dependence on real world events. For example, for a conveyance "`O conveys to A while A is in school`", we add an event "A is in school" before the conveyance.

The majority of failures were due to the presence of complicated conditions that our system does not currently handle (the Complicated column). A few fail due to syntax that is outside what we currently handle (the Syntax column).

Chapter	Total	**Success**	Smoothing	Positive	**Fail**	Complicated	Syntax
1	1	1	0	0	0	0	0
2	31	30	0	0	1	1	0
3	44	40	8	18	4	4	0
4	5	5	0	0	0	0	0
5	38	29	0	0	9	8	2
Total	119	105	8	18	14	13	2

Figure 9. Summary of implemention evaluation on examples from a property law textbook.

In summary, our implementation is capable of handling the majority of examples found in the chapters that are within the scope of our work. The need to add in events to satisfy conditions, as well as the failures due to complicated conditions point to the need to support a richer model of conditions and how they interact with events. This is a primary topic for our future work.

5.3 User Interface

The user interface to the implementation has some additional features that we have not formalized. As an example, recall the extended conveyance from Section 2:

```
1 Olive owns.
2 Olive conveys to Alice for life, then to Bob
    for life until Bob marries, then to Carol.
3 Alice dies.
4 Bob conveys to Dave for life.
5 Bob marries.
```

Figure 10 shows the output of our implementation after executing each line of the example. The output is similar to the representation of the terms we show in Section 2, with some key differences. First, we implement atomic conditions (e.g., "A is alive") in terms of events (e.g., "not(A dies)"). The output of conditions guarding **`while`** and **`if`** subterms refer to the events that affect these conditions, rather than to the natural language originally used to specify the conditions.

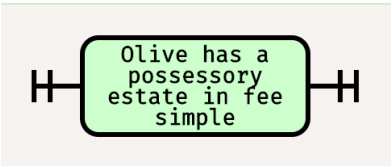
Second, the interface prunes away terms that would be unreachable in practice. In Section 2 the terms have a final **`Atom`** `Olive` subterm, indicating a reversion to the grantor Olive. However, the UI recognizes that Alice's reversionary interest could never be possessory, i.e., the **`Atom`** `Olive` subterm is unreachable, since the preceding preceding interest (**`Atom`** `Carol`) is not guarded by a condition and thus never terminates. Thus, it is safe to prune away the **`Atom`** `Olive` subterm. In fact, this is the correct thing to do from a legal perspective; lawyers would not say that Olive retained an interest in these circumstances. Terms are currently pruned according to a set of heuristics based on the term structure and the properties of the particular conditions involved. We plan to formalize the pruning algorithm in future work.

Finally, the interface annotates each interest term with a human-readable *name* that contains legally relevant information. Much of this name is derived from reasoning about when the interest could become possessory (its *vesting* properties, in legal jargon), and how it might terminate. For example, Bob's interest (in Figure 10b) is initially described as "a remainder in life estate determinable (vested subject to divestment)." It is a *remainder* because it follows Alice's life estate (which terminates at Alice's death). It is *in life estate* because it will be a life estate (i.e., terminable at Bob's death) if it becomes posessory. It is *determinable* because of the added limitation, that Bob must remain unmarried. It is *vested* because all necessary prerequisites to its becoming posessory (i.e., Alice's death) are guaranteed to be satisfied eventually, since Alice will eventually die. And it is *subject to divestment* because Bob's death or marriage could cause it to cease being possessory. But after Alice's death (in 10c), it is labeled as being possessory (rather than a remainder), and in property law the terminology of vesting and divestment only applies to future interests (specifically remainders), not possessory ones.

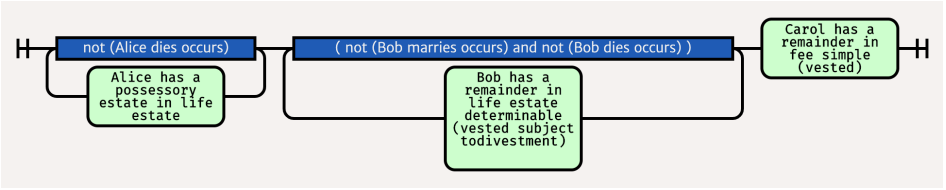
The vesting and naming logic is principled, in the sense that it follows the same logic taught to law students based on the structure of terms and the details of relevant conditions. We have also tested it on examples described in the previous section. However, as noted in Section 4.1, we do not fully model conditions and their relationships with events. Thus the current implementation of the vesting logic relies on heuristics and is limited to a number of common conditions. We plan to formalize the vesting rules in future work, placing the naming logic on surer footing.

6 Related Work

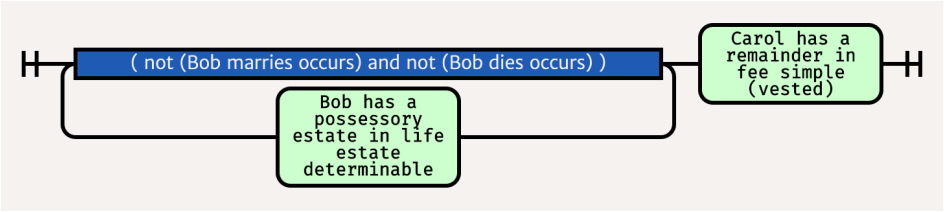
There is a well-established tradition of knowledge representation to enable formally valid reasoning about legal propositions. [McCarty 1976; Sergot et al. 1986]. Our approach is similarly intended to enable deductive reasoning. The principal difference is that we systematically rely on programming language concepts to capture the underlying structure of the legal domain. Our goal is not merely to provide a *knowledge representation* of property law, but also to capture the imperative nature of conveyances as updates to the "state" of the title to a piece of property. To that end, our domain-specific language fits the problem domain more closely than a purely declarative list of facts about property law would.



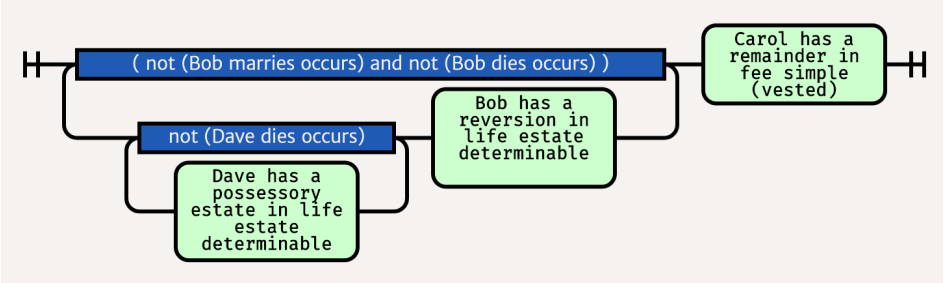
(a) Olive owns.



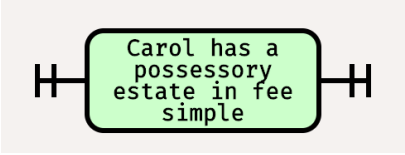
(b) Olive conveys to Alice for life, then to Bob for life until Bob marries, then to Carol.



(c) Alice dies.



(d) Bob conveys to Dave for life.



(e) Bob marries.

Figure 10. User interface output from extended example.

Ours is not the first attempt to formalize some aspect of property law. McCarty [McCarty 2002], following the work of the legal philosopher Wesley Newcomb Hohfeld [Hohfeld 1913], discusses the decomposition of "ownership" into relations between individuals. But this is a much more abstract exercise, meant to clarify the nature of ownership rather than to answer specific legal questions. At the other extreme, Finan and Leyerle [Finan and Leyerle 1987] implemented an expert system in BASIC to walk users through the application of a property law concept known as the *Rule Against Perpetuities*. Their system, however, is essentially a decision tree based on a series of yes-no questions to the user, applicable to one interesting aspect of property law. It has no internal representation of the property interests themselves, nor does it reason more generally about how interests interact. To similar effect is [Becker 1989], which presents a related decision tree as a series of questions for a lawyer to ask, with no attempt at implementation.

The observation that conveyances have a potentially recursive structure, like the outputs of a context-free grammar, is made in a law-school casebook [Merrill and Smith 2017], one of whose authors holds a Ph.D. in linguistics. An earlier version of the point is found in [Lowry 1979]. Previous work by the authors [Basu et al. 2017] describes an abstract syntax for some conveyances that is similar to the concrete syntax described in this paper. However, we did not provide semantics or prove any interesting properties, nor did we explore an implementation in depth. A conveyance interpreter developed by Shawn Bayern, like ours, parses a variety of conveyance patterns, generating graphical repesentations of the resulting interests [Bayern 2010]. The parsing from concrete syntax to abstract syntax trees is based in part on ideas from Bayern's implementation. We extend Bayern's work by developing a model that has semantics as well as syntax: conditions and interests behave in well-defined ways in response to events and conveyances. This approach lets

us take a principled approach to interleaving sequences of conveyances and events, in which the different types of statements interact as they would in real life.

Otherwise, there is little prior work on formal models for property conveyances—especially by comparison with the well-developed body of work formalizing contracts [Hvitved 2010; Pace and Rosner 2009], including approaches that represent contracts using automata [Azzopardi et al. 2016] or functional languages [Jones et al. 2000].

The Legalese project has developed a language for specifying privacy policies that restrict how user data is handled in cloud services, as a step to ensuring that services comply with their privacy policies [Sen et al. 2014]. Recent work on blockchain-based smart contracts has also proposed domain-specific languages for contracts [Accord 2018; Szabo 2002]. The Ethereum blockchain and its associated bytecode programming language has generated a lot of interest in formalizing and verifying properties of smart contracts [Amani et al. 2018; Grech et al. 2018; Tsankov et al. 2018].

Finally, one paper gives an "operational semantics" for a highly abstracted legal system [da Rocha Costa 2015]. It is best understood as an exercise in legal philosophy, working out structural relationships between generic legal concepts.

7 Conclusion & Future Work

In this paper, we have presented the foundation for a formal domain-specific language that describes property transactions. More specifically, we have described:

- A concrete syntax that closely enough approximates a commonly used fragment of written legal English to express literal textbook examples of conveyances.
- A two-step transformation (parsing followed by translation) from the concrete syntax to a simple core calculus with a small number of elementary operators.
- Operational and denotational semantics for the core calculus and a proof of equivalence between them.
- Examples showing that our semantics respects five important and well-known characteristics of actual property conveyances.
- An implementation of our syntax and semantics which correctly analyzes a substantial number of examples from a widely-used legal textbook.

Taken together, these contributions show how an area of law is effectively mechanizable, in the tradition of [McCarty 1976]. It is a small fragment of one field of law, to be sure, but is the essential foundation for formalizing larger portions of that field. We hope that this will be the beginnings of what McCarty calls a "deep conceptual model" of a legal domain [McCarty 1989].

Our work can help apply the power of computing to legal thinking in two ways. One way is forward-looking: it shows how to augment what lawyers currently do by predicting the consequences of different choices, and giving clean ways of accomplishing specific objectives. A lawyer taking this approach would use a tool that modeled the effects of different conveyances until she found one that met her client's objectives; it would then assist in drafting the appropriate documents. Our current implementation is a prototype of such a tool, allowing users to write down conveyances and events and see how they interact. To be more practically useful it would require a better understanding of conditions and events and an improved interface to help users explore their options and understand the effects of their choices.

The second way is backward-looking: it shows how to analyze the current consequences of what lawyers have done in the past. A lawyer taking this approach would use a tool that took a pre-existing natural-language conveyance and generated one or more possible formal representations of it. This approach would require more substantial use of natural-language processing techniques than we have attempted so far. We believe that our formal representations are good candidates for the targets of such natural-language processing.

As noted in Section 5.3, we have formalized a subset of the features of our current implementation. Most significantly, our implementation currently also prunes away unreachable interests and decorates interests with names according to the property-law *naming* rules. In future work, we will describe in formal detail these naming rules.

There are also a significant number of additional property doctrines that we plan to incorporate into our framework. These include the doctrines of *the destructability of contingent remainders*, *vesting*, *merger*, *the Rule in Shelley's Case*, *the Doctrine of Worthier Title*, and *the Rule Against Perpetutites*. An important theme tying together many of these doctrines is that they depend on determining whether a future interest is *vested*—roughly, guaranteed to become posseessory at some future time. This concept depends on reasoning about future events, and on understanding conditions in detail. Thus, future work will need a better way to model conditions and their relations to events, and properly integrate reasoning about conditions with our current reasoning about language terms. We suspect that we will need to leverage existing work on temporal logics and their related automata-theoretic models to achieve this. Other fruitful directions for extending our framework would be to add representations for shared simultaneous ownership (e.g. *joint tenancies*) and to model the effects of state *recording acts* for adjudicating conflicting conveyances.

A Equivalence Proof

This section gives a proof of Theorem 4.1.

Proof. By induction on $|\bar{s}_0|$, $|\bar{s}_1|$ and t. Each use of the inductive hypothesis is either on sub-sequence of $\bar{s}_0$ or $\bar{s}_1$ or a sub-term of t. The proof is nearly a structural induction on t, but the case for Atom γ requires an inductive hypothesis for a longer t due to substitution. We analyze several cases.

Case: t = **Bottom:**

Note that for all $\bar{s}$ we have:

$$\langle \text{Bottom} \rangle_{\bar{s}} = \text{Bottom} = \Delta_{\bar{s}}(\text{Bottom})$$

Using this fact, we calculate as follows:

$$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{Bottom} \rangle_{\bar{s}_0}) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{Bottom})) \\ = {} & E(\text{Bottom}) \\ = {} & \bot \\ = {} & [\![\text{Bottom}]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

obtaining the required equality, which finishes the case.

Case t = **Atom** γ**:**

We anlayze several sub-cases.

- Suppose that $\bar{s}_0 \in S^*_\gamma$ and $\bar{s}_1 \in S^*_\gamma$. Note that for all $\bar{s} \in S^*_\gamma$ we have:

  $$\langle \text{Atom } \gamma \rangle_{\bar{s}} = \text{Atom } \gamma = \Delta_{\bar{s}}(\text{Atom } \gamma)$$

  Using this fact, we calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{Atom } \gamma \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{Atom } \gamma)) \\ = {} & E(\text{Atom } \gamma) \\ = {} & \gamma \\ = {} & [\![\text{Atom } \gamma]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.
- Suppose that $\bar{s}_0 = \bar{s}'_0(\gamma, t')\bar{s}''_0$ where $\bar{s}'_0 \in S^*_\gamma$. Note that:

  $$\langle \text{Atom } \gamma \rangle_{\bar{s}'_0(\gamma,t')\bar{s}''_0} = \langle t' \,;\, \text{Atom } \gamma \rangle_{\bar{s}'_0\bar{s}''_0}$$

  Moreover, by uniqueness of interests, we also have

  $$\langle \bar{s}_1 \rangle_{\bar{s}'_0(\gamma,t)\bar{s}''_0} = \langle \bar{s}_1 \rangle_{\bar{s}'_0\bar{s}''_0}$$

  Using these facts, we calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{Atom } \gamma \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}'_0(\gamma,t')\bar{s}''_0}}(\langle \text{Atom } \gamma \rangle_{\bar{s}'_0(\gamma,t')\bar{s}''_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}'_0(\gamma,t')\bar{s}''_0}}(\langle t' \,;\, \text{Atom } \gamma \rangle_{\bar{s}'_0\bar{s}''_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}'_0\bar{s}''_0}}(\langle t' \,;\, \text{Atom } \gamma \rangle_{\bar{s}'_0\bar{s}''_0})) \\ = {} & [\![t' \,;\, \text{Atom } \gamma]\!](\bar{s}'_0\bar{s}''_0, \bar{s}_1) \\ = {} & [\![\text{Atom } \gamma]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.
- Suppose that $\bar{s}_0 \in S^*_\gamma$, but $\bar{s}_1 = \bar{s}'_1(\gamma, t')\bar{s}''_1$ where $\bar{s}'_1 \in S^*_\gamma$. Note that:

  $$\Delta_{\langle \bar{s}'_1(\gamma,t')\bar{s}''_1 \rangle_{\bar{s}_0}}(\text{Atom } \gamma) = \Delta_{\langle \bar{s}''_1 \rangle_{\bar{s}_0\bar{s}'_1}}(\langle t' \rangle_{\bar{s}_0\bar{s}'_1} \,;\, \text{Atom } \gamma)$$

  Using this fact, we calculate as follows:

  $$\begin{aligned} & = E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{Atom } \gamma \rangle_{\bar{s}_0})) \\ & = E(\Delta_{\langle \bar{s}'_1(\gamma,t')\bar{s}''_1 \rangle_{\bar{s}_0}}(\langle \text{Atom } \gamma \rangle_{\bar{s}_0})) \\ & = E(\Delta_{\langle \bar{s}'_1(\gamma,t')\bar{s}''_1 \rangle_{\bar{s}_0}}(\text{Atom } \gamma)) \\ & = E(\Delta_{\langle \bar{s}''_1 \rangle_{\bar{s}_0\bar{s}'_1}}(\langle t' \rangle_{\bar{s}_0\bar{s}'_1} \,;\, \text{Atom } \gamma)) \\ & = E(\Delta_{\langle \bar{s}''_1 \rangle_{\bar{s}_0\bar{s}'_1}}(\langle t' \,;\, \text{Atom } \gamma \rangle_{\bar{s}_0\bar{s}'_1})) \\ & = [\![t' \,;\, \text{Atom } \gamma]\!](\bar{s}_0\bar{s}'_1, \bar{s}''_1) \\ & = [\![\text{Atom } \gamma]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case and the case.

Case t = **If**(c, t)**:**

We analyze several sub-cases.

- Suppose that $\bar{s}_0 \models c$. Note that $\models \langle c \rangle_{\bar{s}_0}$. Using this fact, we calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{If}(c, t) \rangle_{\bar{s}_0}) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{If}(\langle c \rangle_{\bar{s}_0}, \langle t \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0}) \\ = {} & [\![t]\!](\bar{s}_0, \bar{s}_1) \\ = {} & [\![\text{If}(c, t)]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.
- On the other hand, suppose that $\bar{s_0} \not\models c$. Note that $\not\models \langle c \rangle_{\bar{s}_0}$. We calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{If}(c, t) \rangle_{\bar{s}_0}) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{If}(\langle c \rangle_{\bar{s}_0}, \langle t \rangle_{\bar{s}_0})) \\ = {} & E(\text{Bottom}) \\ = {} & \bot \\ = {} & [\![\text{If}(c, t)]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.

Case t = **While**(c, t)**:**

We analyze several sub-cases.

- Suppose that $C[\![c]\!](\bar{s}_0, \bar{s}_1)$ and $\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0}) \neq \text{Bottom}$. Note that $\models \langle c \rangle_{\bar{s}_0}$. We calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{While}(c, t) \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{While}(\langle c \rangle_{\bar{s}_0}, \langle t \rangle_{\bar{s}_0}))) \\ = {} & E(\text{While}(\langle c \rangle_{\bar{s}_0\bar{s}_1}, \Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0}))) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0})) \\ = {} & [\![t]\!](\bar{s}_0, \bar{s}_1) \\ = {} & [\![\text{While}(c, t)]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.
- Suppose that $C[\![c]\!](\bar{s}_0, \bar{s}_1)$ and $\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0}) = \text{Bottom}$. We calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{While}(c, t) \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{While}(\langle c \rangle_{\bar{s}_0}, \langle t \rangle_{\bar{s}_0}))) \\ = {} & E(\text{While}(\langle c \rangle_{\bar{s}_0\bar{s}_1}, \Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0}))) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t \rangle_{\bar{s}_0})) \\ = {} & E(\text{Bottom}) \\ = {} & \bot \\ = {} & [\![\text{While}(c, t)]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.
- Suppose that $\neg C[\![c]\!](\bar{s}_0, \bar{s}_1)$. Note that $\not\models \langle c \rangle_{\bar{s}_0}$. We calculate as follows:

  $$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle \text{While}(c, t) \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\text{While}(\langle c \rangle_{\bar{s}_0}, \langle t \rangle_{\bar{s}_0}))) \\ = {} & E(\text{Bottom}) \\ = {} & \bot \\ = {} & [\![\text{While}(c, t)]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case and the case.

Case $t = t_1 \,;\, t_2$**:**

We analyze several sub-cases.

- Suppose that $\exists \min \bar{s}_1' : \bar{s}_1 = \bar{s}_1'\bar{s}_1'' \wedge \Delta_{\langle \bar{s}_1' \rangle_{\bar{s}_0}}(\langle t_1 \rangle_{\bar{s}_0}) =$ Bottom, then $\bar{s}_1' \in B(T([\![t_1]\!]_\perp(\bar{s}_0)))$. We calculate as follows:

$$\begin{aligned} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t_1 \,;\, t_2 \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t_1 \rangle_{\bar{s}_0} \,;\, \langle t_2 \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1'\bar{s}_1'' \rangle_{\bar{s}_0}}(\langle t_1 \rangle_{\bar{s}_0} \,;\, \langle t_2 \rangle_{\bar{s}_0})) \\ = {} & E(\Delta_{\langle \bar{s}_1'' \rangle_{\bar{s}_0\bar{s}_1'}}(\langle t_2 \rangle_{\bar{s}_0\bar{s}_1'})) \\ = {} & [\![t_2]\!](\bar{s}_0\bar{s}_1', \bar{s}_1'') \\ = {} & [\![t_1 \,;\, t_2]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case.
- Suppose that no such $\bar{s}_1'$ exists. Then there is no $\bar{s}_1' \leq \bar{s}_1$ such that $\bar{s}_1' \in B(T([\![t_1]\!]_\perp(\bar{s}_0)))$. We calculate as follows:

$$\begin{aligned} & = E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t_1 \,;\, t_2 \rangle_{\bar{s}_0})) \\ & = E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t_1 \rangle_{\bar{s}_0} \,;\, \langle t_2 \rangle_{\bar{s}_0})) \\ & = E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t_1 \rangle_{\bar{s}_0}) \,;\, \langle t_2 \rangle_{\bar{s}_0\bar{s}_1}) \\ & = E(\Delta_{\langle \bar{s}_1 \rangle_{\bar{s}_0}}(\langle t_1 \rangle_{\bar{s}_0})) \\ & = [\![t_1]\!](\bar{s}_0, \bar{s}_1) \\ & = [\![t_1 \,;\, t_2]\!](\bar{s}_0, \bar{s}_1) \end{aligned}$$

  obtaining the required equality, finishing the sub-case, the case, and the proof. □

Acknowledgments

We wish to thank the Onward! reviewers for helpful comments and suggestions. Thanks to Michael Greenberg and Stephen Chong for providing valuable feedback on the semantics. This work was supported by the National Science Foundation under grant CNS-143972.

References

Accord. 2018. Introduction to Ergo, Accord Project. Retrieved April 29,2019 from https://docs.accordproject.org/docs/ergo.html

Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. 2018. Towards Verifying Ethereum Smart Contract Bytecode in Isabelle/HOL. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, New York, NY, USA, 66–77.

American Law Institute. 1936. *Restatement of Property.*

American Law Institute. 1979. *Restatement (Second) of Torts.*

Valentin M. Antimirov. 1996. Partial Derivatives of Regular Expressions and Finite Automaton Constructions. *Theor. Comput. Sci.* 155, 2 (1996), 291–319. https://doi.org/10.1016/0304-3975(95)00182-4

Holger Arnold and Max Mouratov. 2008. MParser, a simple monadic parser combinator library. Retrieved April 29,2019 from https://github.com/murmour/mparser/

Shaun Azzopardi, Gordon Pace, Fernando Schapachnik, and Gerardo Schneider. 2016. Contract automata: An operational view of contracts between interactive parties. *Artificial Intelligence and Law* 24 (09 2016). https://doi.org/10.1007/s10506-016-9185-2

Shrutarshi Basu, James Grimmelmann, and Nate Foster. 2017. Property Law as a Programming Language. Presentation. In *Domain Specific Languages Design and Implementation*. ACM.

Shawn J. Bayern. 2010. A Formal System for Analyzing Conveyances of Property Under the Common Law. In *Proceedings of the 2010 Conference on Legal Knowledge and Information Systems: JURIX 2010: The Twenty-Third Annual Conference*. IOS Press, Amsterdam, The Netherlands, 139–142. http://dl.acm.org/citation.cfm?id=1940559.1940579

David M Becker. 1989. A Methodology for Solving Perpetuites Problems under the Common Law Rule: A Step-by-Step Process that Carefully Identifies All Testing Lives in Being. *Wash. ULQ* 67 (1989), 949.

Janusz A Brzozowski. 1964. Derivatives of regular expressions. In *Journal of the ACM*. Citeseer.

Antônio Carlos da Rocha Costa. 2015. Situated legal systems and their operational semantics. *Artificial Intelligence and Law* 23, 1 (01 Mar 2015), 43–102. https://doi.org/10.1007/s10506-015-9164-z

Linda Edwards. 2009. *Estates in land and future interests : a step-by-step guide*. Wolters Kluwer Law & Business Aspen Publishers, Austin New York, NY.

John P Finan and Albert H Leyerle. 1987. The Perp Rule Program: Computerizing the Rule Against Perpetuities. *Jurimetrics* 28 (1987), 317–335.

Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts. In *Proceedings of the 31st Annual ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems, Languages, and Applications*. ACM, New York, NY, USA.

James Grimmelmann. 2017. Real + Imaginary = Complex: Toward a Better Property Course. *Journal of Legal Education* 66 (2017), 930–955.

Wesley Newcomb Hohfeld. 1913. Some fundamental legal conceptions as applied in judicial reasoning. *Yale Law Journal* 23 (1913), 16–59.

Tom Hvitved. 2010. A survey of formal languages for contracts. *Formal Languages and Analysis of Contract-Oriented Software* (2010), 29–32.

S Peyton Jones, Jean-Marc Eber, and Julian Seward. 2000. Composing contracts: an adventure in financial engineering. *ACM SIGPLAN Notices* 35, 9 (2000), 280–292.

Sarah B Lawsky. 2016. Formalizing the Code. *Tax Law Review* 70 (2016), 377.

Houston Putnam Lowry. 1979. Normalization: A Revolutionary Approach. *Jurimetrics* 20, 2 (1979), 140–144.

L Thorne McCarty. 1976. Reflections on TAXMAN: An experiment in artificial intelligence and legal reasoning. *Harv. L. Rev.* 90 (1976), 837.

L Thorne McCarty. 1989. A language for legal Discourse I. basic features. In *ICAIL*, Vol. 89. 180–189.

L Thorne McCarty. 2002. Ownership: A case study in the representation of legal concepts. *Artificial Intelligence and Law* 10, 1-3 (2002), 135–161.

Thomas W. Merrill and Henry E. Smith. 2017. *Property : principles and policies*. Foundation Press, St. Paul, MN.

Gordon J Pace and Michael Rosner. 2009. A controlled language for the specification of contracts. In *International Workshop on Controlled Natural Language*. 226–245.

Shayak Sen, Saikat Guha, Anupam Datta, Sriram K. Rajamani, Janice Y. Tsai, and Jeannette M. Wing. 2014. Bootstrapping Privacy Compliance in Big Data Systems. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 327–342. https://doi.org/10.1109/SP.2014.28

Marek J Sergot, Fariba Sadri, Robert A Kowalski, Frank Kriwaczek, Peter Hammond, and H Terese Cory. 1986. The British Nationality Act as a logic program. *Commun. ACM* 29, 5 (1986), 370–386.

Nick Szabo. 2002. A Formal Language for Analyzing Contracts. Retrieved April 29,2019 from http://nakamotoinstitute.org/contract-language/

Petar Tsankov, Andrei Marian Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin T. Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 67–82. https://doi.org/10.1145/3243734.3243780