

Property Law as a Programming Language

Shrutarshi Basu, Nate Foster, & James Grimmelman

Internet Law Works in Progress

March 24, 2018

Demo

What was that?

- *Not* the first conveyance interpreter
 - Shawn Bayern wrote one in 2010
- *Not* the first legal AI
 - Nowhere near the most sophisticated
- But it *is* a simple, clean, and rigorous implementation of future interests
 - The “how” is more important than the “what”

Big idea:

programming language theory

- Future interests feel computerizable because:
 - They're highly rule-bound
 - Those rules have a recursive structure
- You know what else has a recursive set of rules that define valid expressions and their effects?
 - *Programming languages.*
- Thus, we define syntax and semantics for a “domain-specific language” in which conveyances are programs

Key concepts

- A *grammar* that defines the set of possible conveyances in terms of their *abstract syntax trees* (ASTs)
- A *parser* that turns a specific conveyance into their corresponding ASTs
- A simplified *core language* that is easy to reason about
- A *translator* from turns ASTs into terms in the core language
- *Semantics* that specify unambiguously how the core language functions

Grammar (simplified)

Person → Alice, Bob, ...

Duration → for life
→ and her/his heirs

Grant → to Person Duration

Grants → Grant
→ Grant, then Grants

Parsing (pt. 1)

to Alice for life, then to Bob and his heirs
to Alice **Duration**, then to Bob and his heirs

to Alice Duration, then to Bob and his heirs
to **Person** Duration, then to Bob and his heirs

Person Duration, then to Bob and his heirs
Grant, then to Bob and his heirs

Parsing (pt. 2)

Grant, then to Bob and his heirs

Grant, then to Bob **Duration**

Grant, then to Bob Duration

Grant, then to **Person** Duration

Grant, then to Person Duration

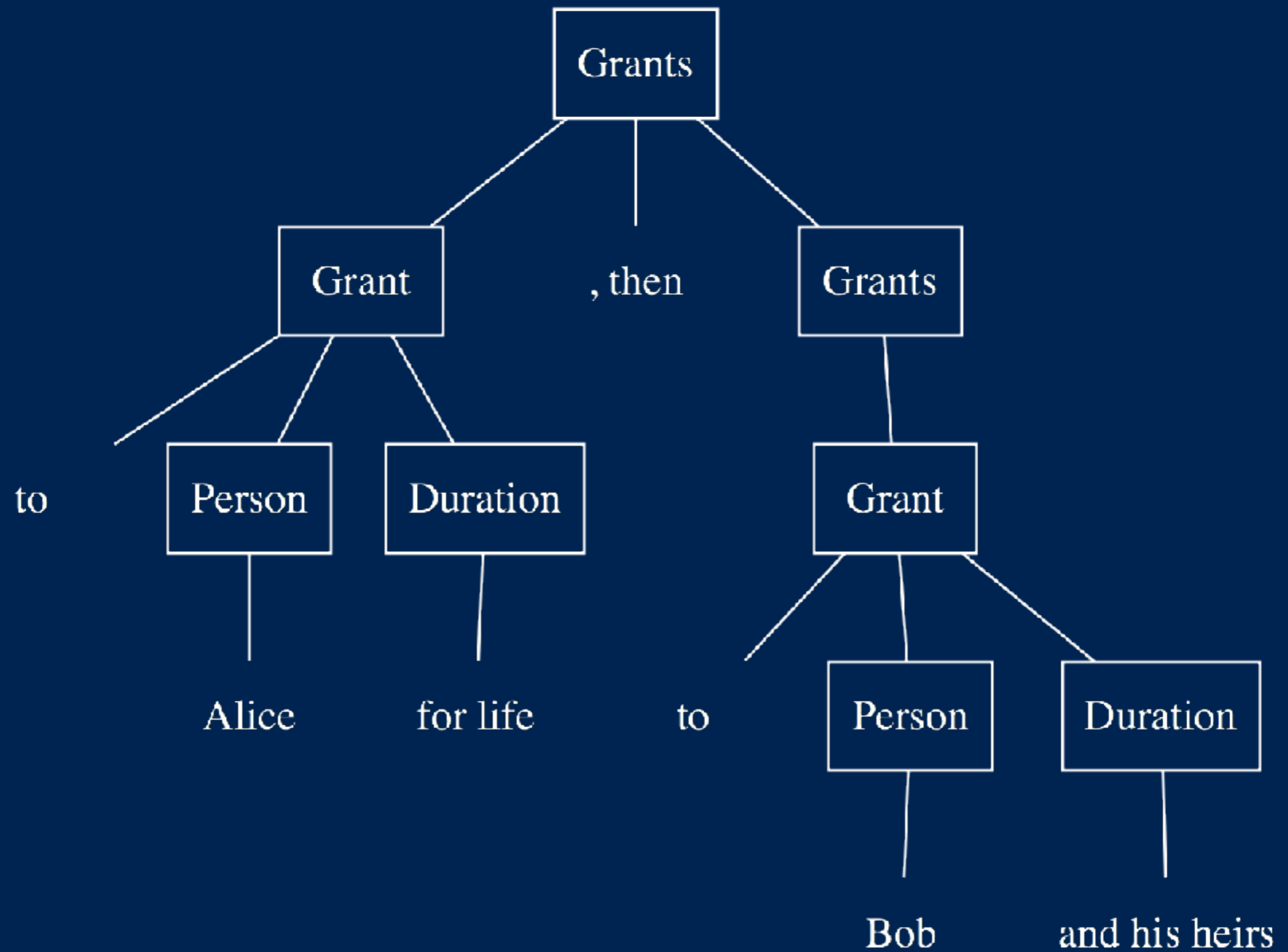
Grant, then **Grant**

Parsing, pt. 3

Grant, then Grant
Grant, then **Grants**

Grant, then Grant
Grants

AST



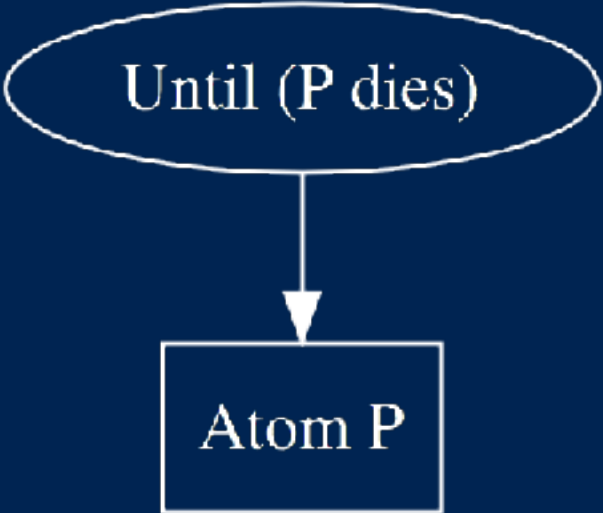
Core language (simplified)


$p \in \text{Persons}$
 $c \in \text{Conditions}$
 $t, t1, t2, \dots \in \text{Terms}$

Term ::= Owns p
 | Until c t
 | Then t1 t2

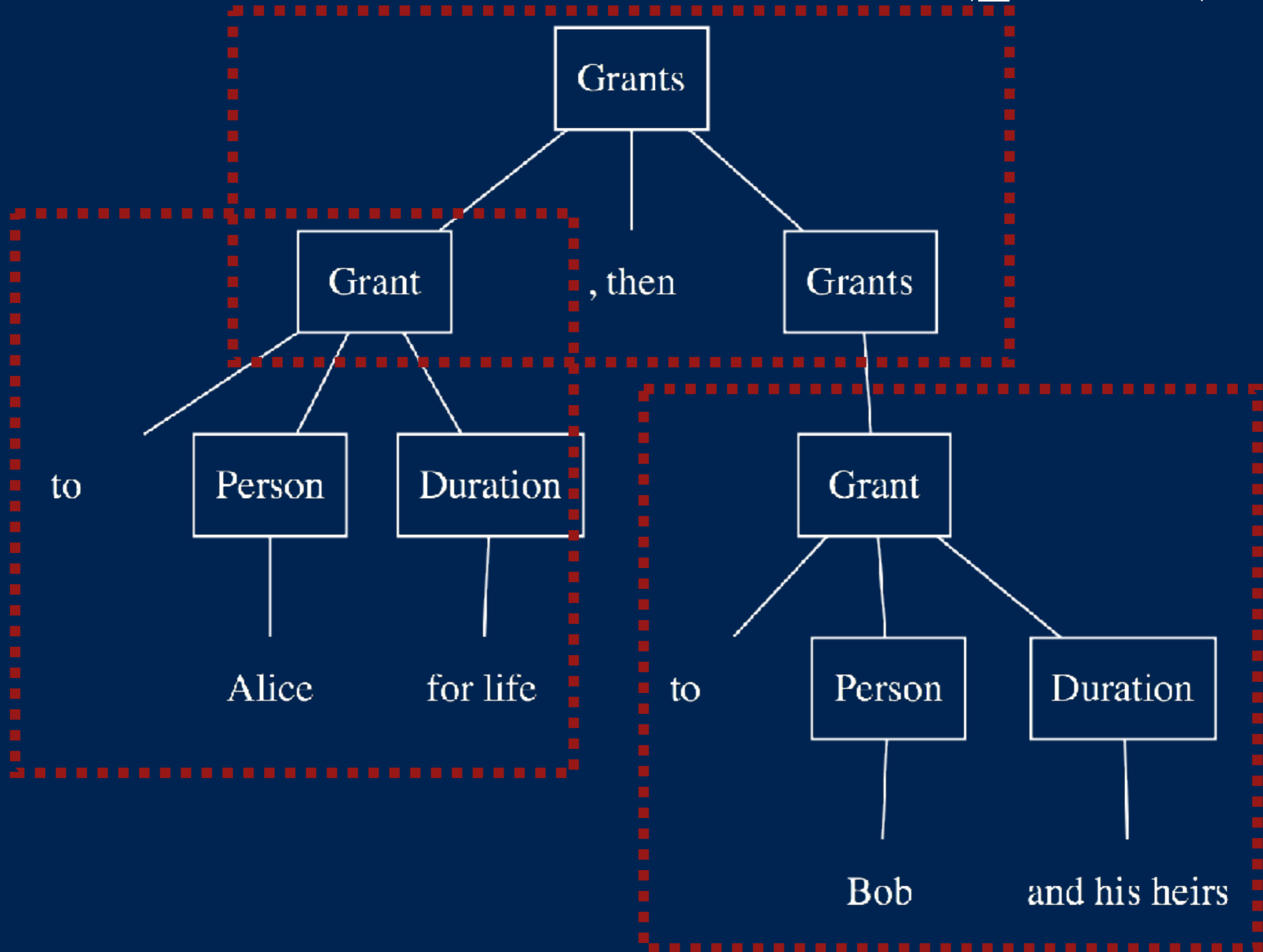
Translation from ASTs to core language terms (pt. 1)

[[Grant(P, absolute)]] = 

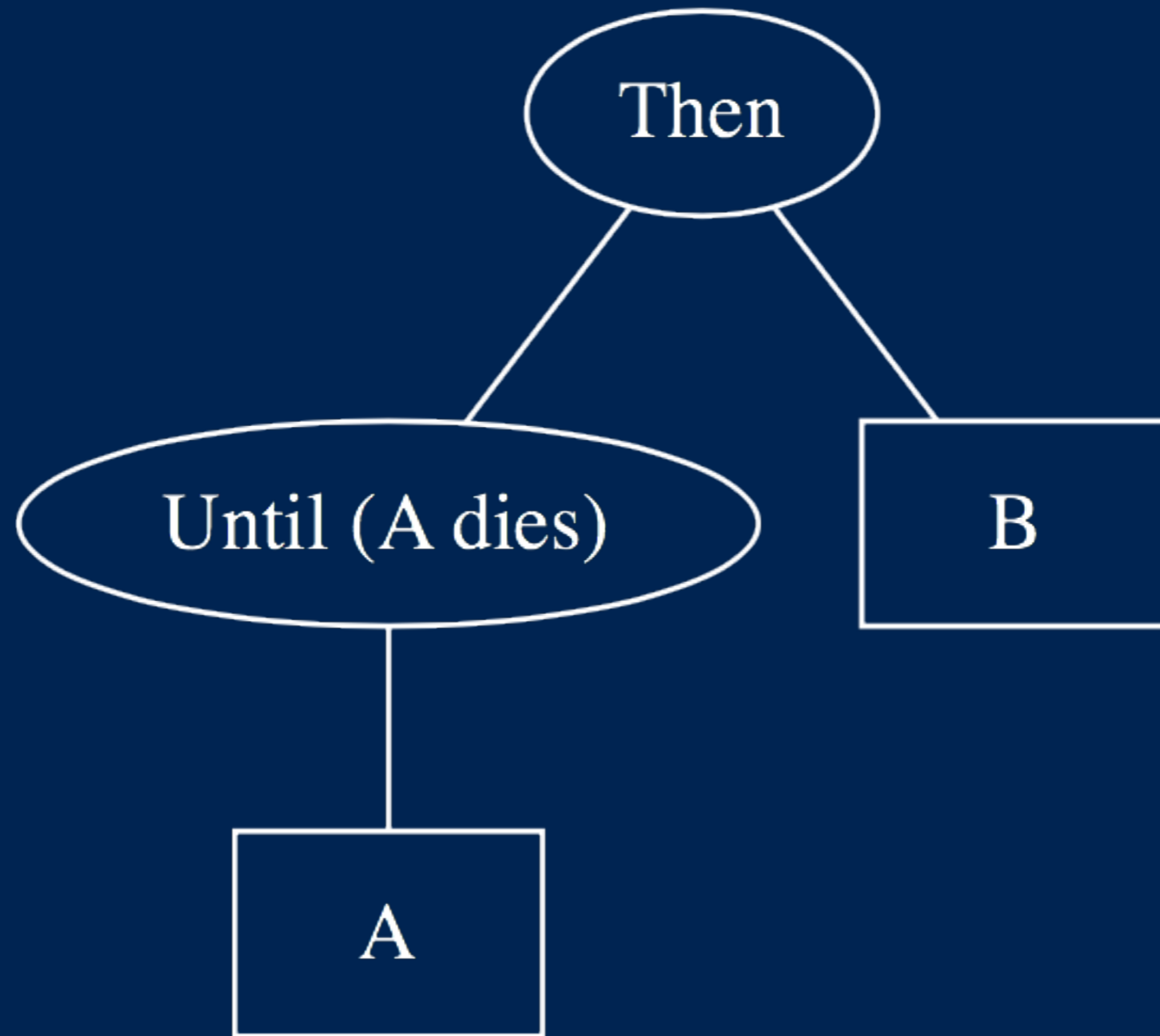
[[Grant(P, life)]] = 

[[Grants(G1, G2)]] = 

Translation (pt. 2)



Core language term



Semantics

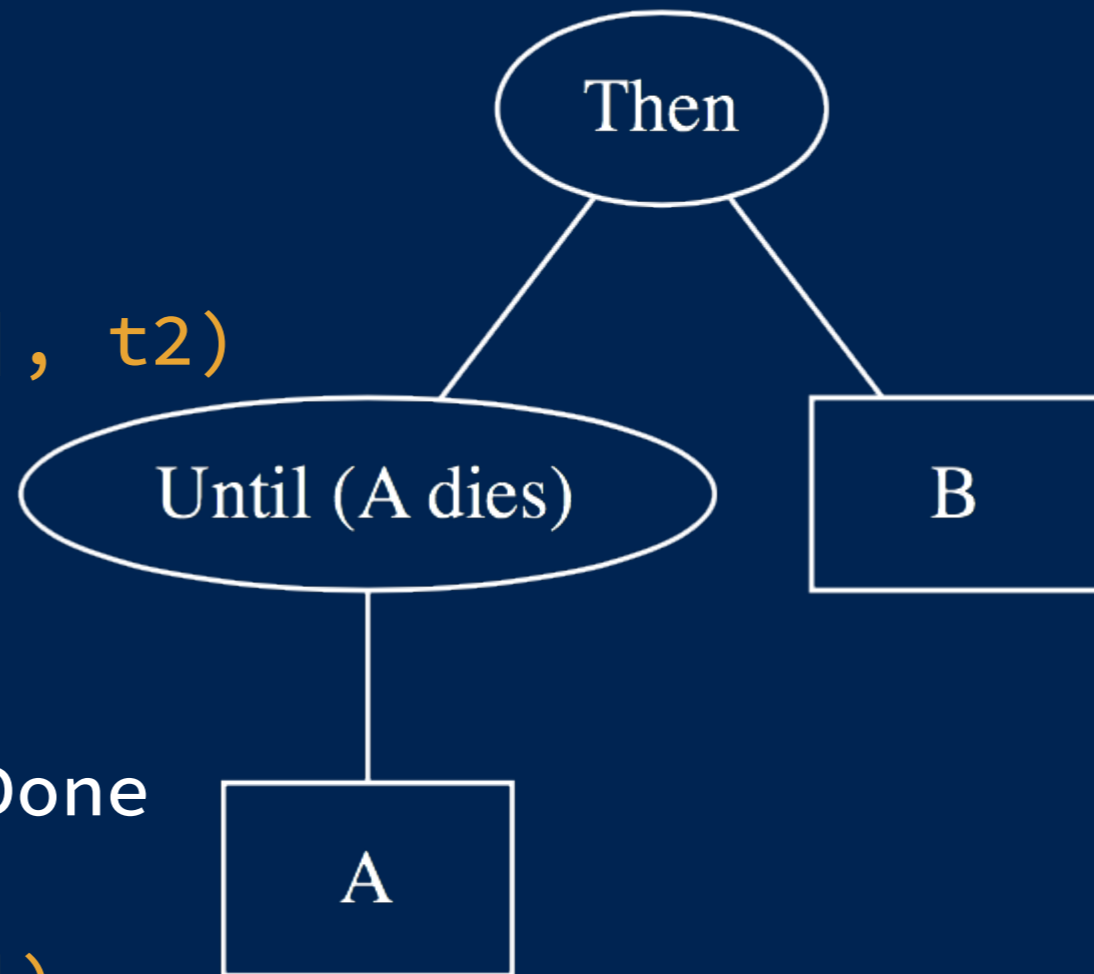
$h[[\text{Owns}(p)]] = \text{Owns}(p)$

$h[[\text{Until}(c,t)]] =$
if $c(h)$ or $h[[t]] = \text{Done}$
then Done
else $\text{Until}(h[[t]])$

$h[[\text{Then}(t_1,t_2)]] =$
if $h[[t_1]] = \text{Done}$
then $h[[t_2]]$
else $\text{Then}(h[[t_1]], t_2)$

Semantics (example): before A dies

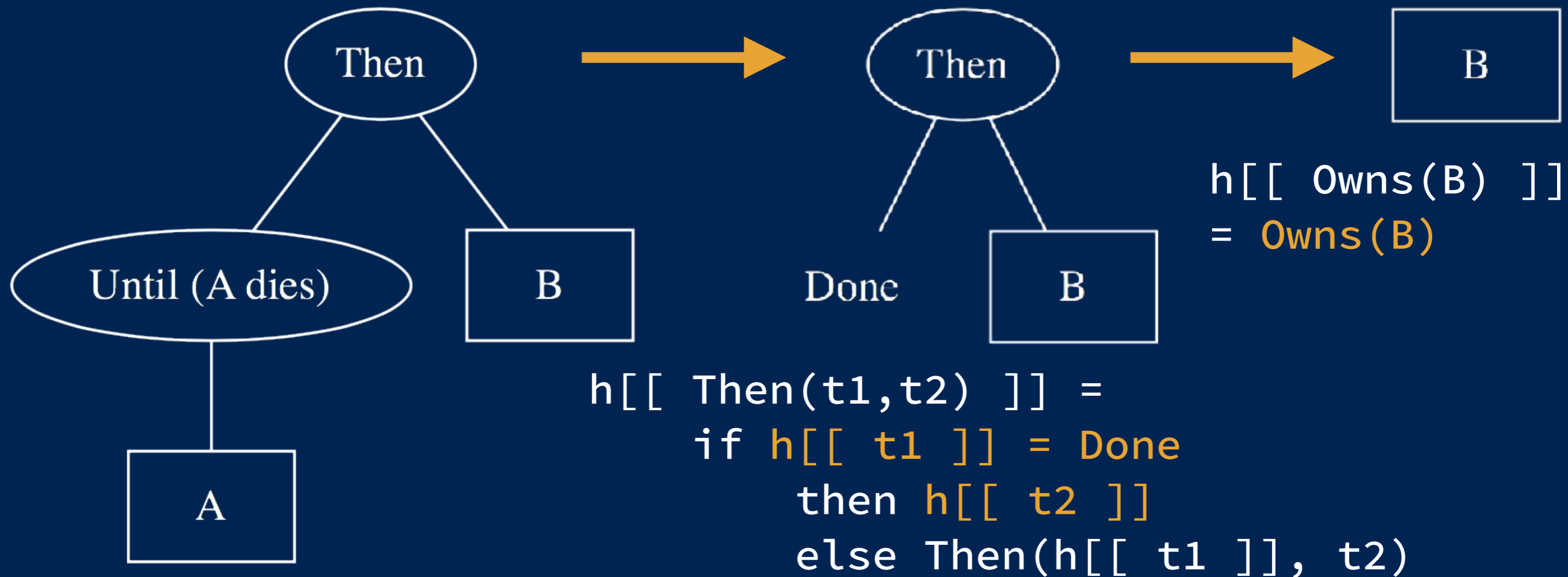
```
h[[ Then(t1,t2) ]] =  
  if h[[ t1 ]] = Done  
    then h[[ t2 ]]  
    else Then(h[[ t1 ]], t2)
```



```
h[[ Until(c,t) ]] =  
  if c(h) or h[[ t ]] = Done  
    then Done  
    else Until(h[[ t ]])
```

```
h[[ Owns(A) ]] = Owns(A)
```


Semantics (example): after A dies



$h[[\text{Then}(t_1, t_2)]] =$
 if $h[[t_1]] = \text{Done}$
 then $h[[t_2]]$
 else $\text{Then}(h[[t_1]], t_2)$

$h[[\text{Until}(c, t)]] =$
 if $c(h)$ or $h[[t]] = \text{Done}$
 then Done
 else $\text{Until}(h[[t]])$

Extensions and *future work*

- More quanta: **term of years**, *fee tail*, etc.
- Implicit syntax: **implied reversions**, **fee simple by default**, etc.
- Limitations: **executory interests**, **special limitations**, **conditions subsequent**, etc.
- **Conditions precedent**
- More events and conditions, e.g. “*A survives B*”
- **Naming interests** (incl. **vesting**)
- Other doctrines: *merger*, **destructibility of contingent remainders**, *rule against perpetuities*, etc.

Takeaways pt 1: lessons for property law

- The basic grammar of future interests really is simple and clean ...
- ... but not quite as simple and clean as it seems
- Very few property teachers (myself included) really understand all of the nuances of the First Restatement's rules
- There are ambiguities everywhere!

Takeaways pt. 2: why programming languages?

- Close fit: legal problem & technical solution
- PL techniques pushed us toward simple and elegant descriptions of property law
- Complementary perspective on legal *language*
- Applications: teaching tools, smart contracts, legal AI, law reform, etc.

Questions?