Software Interpretation for Lawyers

James Grimmelmann

March 6, 2019

In this talk

- Draw out the similarities and differences in:
  - … how legal actors interpret legal texts
  - … how computers interpret programs
- Motivating example: unauthorized access

Motivation

Compare:

"You are not allowed to access any files in the `/private/` directory."

```
User-agent: *
Disallow: /private/
```

Compare:

"You are not alloqed to access any files in the `/private/` directory."

```
User-agent: *
Disalloq: /private/
```

Intuition

- In a natural-language legal text, a typo like “alloqed” for “allowed” would typically be trivially ignored
- In a formal-language program, a typo like `Disalloq` for `Disallow` poses a harder question:
  - Correct it to `Disallow`, like a human would?
  - Or leave it as `Disalloq`, like a computer would?
- Goal: a framework for thinking about such questions

Legal speech acts

Software with legal effects

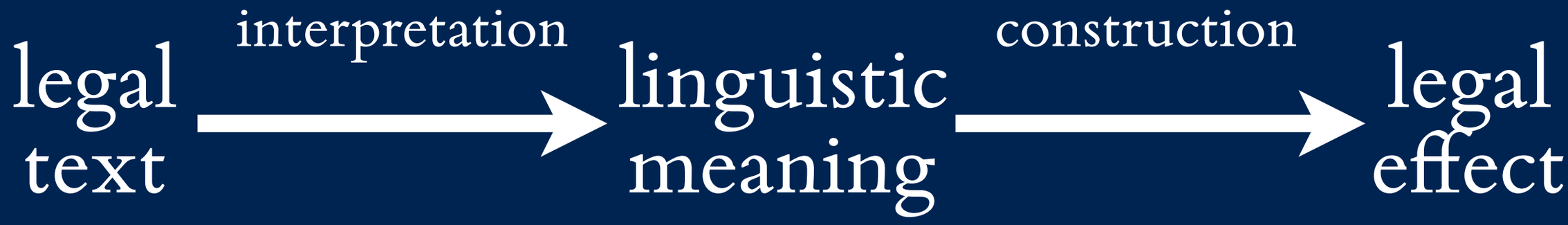
- Software can convey permission (to use it)
- Obvious analogies: statutes, licenses, etc.
- These have their own legal interpretive rules
- *What are the interpretive rules for software?*

Legal speech acts

- "Be it hereby enacted that …" is a *speech act*
  - It has the *illocutionary force* of changing the law (and possibly also of commanding subjects to comply and officials to act.)
- Other legal speech acts: contracts, wills, ToS, etc.
  - They have their own illocutionary forces
- Their *felicity conditions* are that (1) the speaker must have appropriate authority, and (2) the speaker must follow the correct formalities

Interpretation and construction

- It is helpful to divide legal interpretation into:
  - *Interpretation* — the process of determining the linguistic meaning of the text
  - *Construction* — the process of determining the legal effect, given the linguistic meaning
- Both are complex processes. Interpretation can eliminate some linguistic ambiguity, and construction must clean up the rest


legal text
interpretation
linguistic meaning
construction
legal effect

“not alloqed” —interpretation→ not allowed —construction→ ~~/private/~~

Software speech acts

Software speech acts

- `print(2+2)` is also a kind of speech act
- When uttered to a Python interpreter, it causes the computer to display `4`
- We could talk about this mechanistically, deny that the computer understands anything, and deny that communication is taking place
- But this overlooks the ways in which `print(2+2)` is *linguistically* meaningful

Examples

- *E.g., Bernstein v. DoJ*: software can be First-Amendment-covered speech
- *E.g., Computer Associates v. Altai*: software can be copyrightable
- Neither of these cases is intelligible if software is inherently only a functional artifact
- For better or for worse, we program computers with words that have meaning to humanns


interpretation
program
text
functional
effect

interpretation

Program meaning

Who is the interpreter?

- Legal texts are addressed to *people*: citizens, counterparties, guests, and especially judges
  - They mean what they mean to people
- Programs are addressed to *computers*: they consists of a series of commands to execute
  - Do they mean (only) what they cause computers to do?

Program meaning

- A family of theories that a program's meaning is determined by what it causes a computer to do
- A naïve version would look to to a specific physical computer an actual execution at a specific time+place
    - This is obviously insufficient, because it ignores that actual computers can and do malfunction
- A more sophisticated version would look to an idealized, abstracted, correctly functioning computer

Where does program meaning come from?

- What does `2**2` mean in a programming language?
- Three answers:
  - Use a program: a *reference implementation* whose behavior is by stipulation treated as correct
  - Use natural language: a *specification* that defines the behavior of a correct implementation
  - Use mathematics: a *formal semantics* that identifies programs with abstract entities

More questions

- Where do specifications and semantics come from?
  - Some people got together to define them
- What language are we running?
  - “Python” 2.7 is different from “Python” 3.6
  - `print 2+2` is a valid program only in the former
- These questions can be answered only by reference to a community of programmers and users

Program meaning = extreme literal meaning

- Programming languages have determinate syntax, semantics, and pragmatics:
  - No internal ambiguity or vagueness
    - No implicature
    - No reference to the outside world
- Ambiguity is pushed upstream from the *program*'s semantics into the *language*'s

Other meanings

But wait, there's more!

- Programmers and users routinely act in ways that show they consider program meaning inadequate for their purposes
  - E.g., the very idea of a "bug" presupposes that program meaning might fail to reflect a programmer's communicative intent
  - E.g., the CFAA would be incoherent if program meaning determined authorization

Programmer meaning

- A family of theories that a program's meaning is what a programmer would believe the program is attempting to do
- Obviously comes in many variants corresponding to whose perspective one adopts (e.g., author vs. reasonable programmer) and what contextual information one looks to (e.g. documentation)
- A reasonable programmer might understand that `pint(2+2)` is a *buggy* program to print `4`
- The communication to readers of the code succeeds, even if the program itself fails to execute

Incidental meaning

- In Python, lines starting with # are ignored
  - Programmers use comments to document their work, for themselves and others
  - Or to make jokes, etc.
- This is *incidental meaning*: it is independent of what the program does

```
from itertools import repeat
for feet in [3,3,2,2,3]:
      print " ".join("DA-DA-DUM"
      for dummy in [None]
for foot in repeat("metric", feet))
```

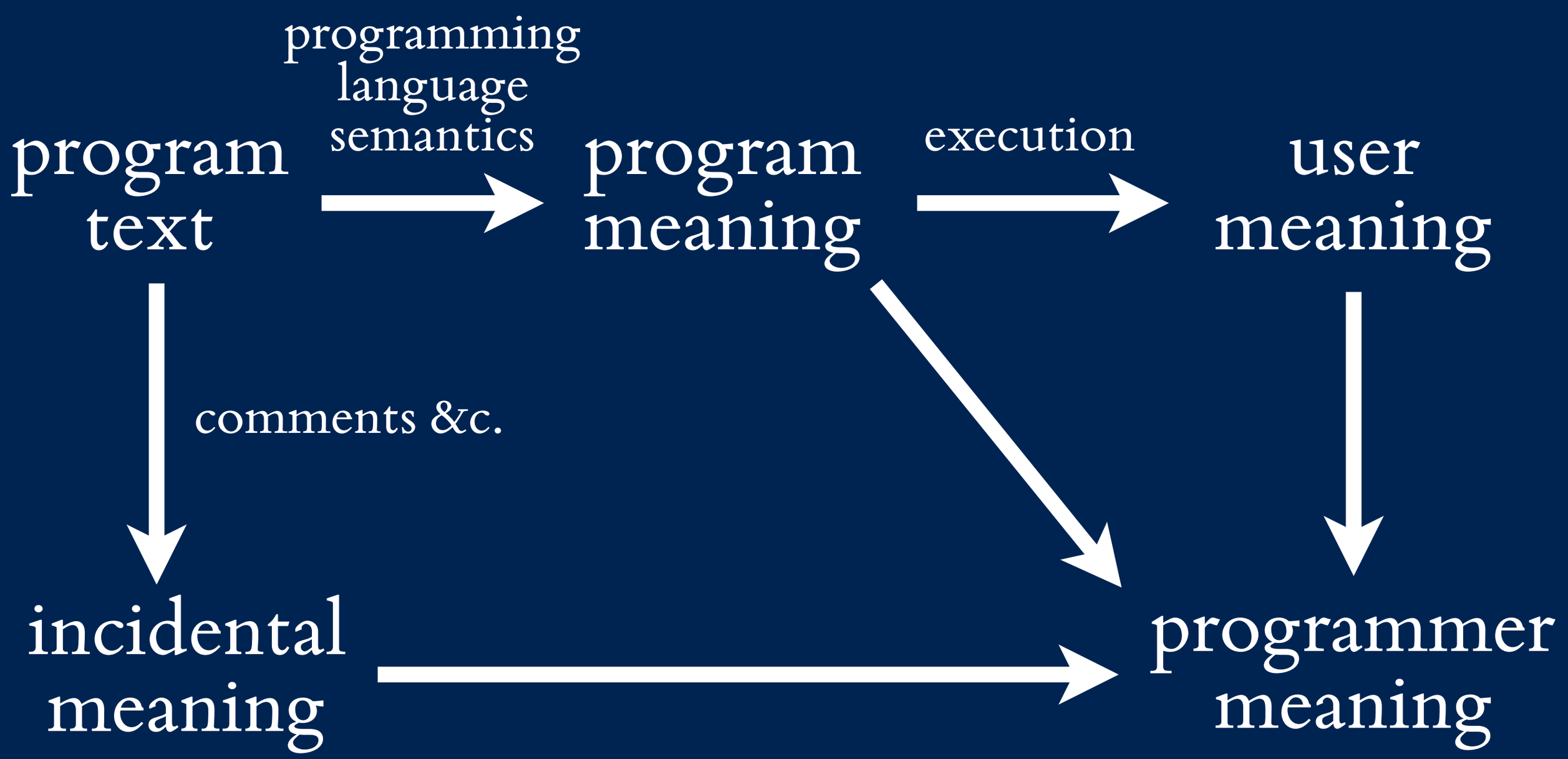
```
DA-DA-DUM DA-DA-DUM DA-DA-DUM
DA-DA-DUM DA-DA-DUM DA-DA-DUM
DA-DA-DUM DA-DA-DUM
DA-DA-DUM DA-DA-DUM
DA-DA-DUM DA-DA-DUM DA-DA-DUM
```

```
                                  /*
                                 +
                                +
                               +
                              +
                              +
                              [          >i>n[t
                               */   #include<stdio.h>
                          /*2w0,1m2,]_<n+a m+o>r>i>=>(['0n1'0)1;
                        */int/**/main(int/**/n,char**m){FILE*p,*q;int        A,k,a,r,i/*
                      #uinndcelfu_dset<rsitcdti_oa.nhs>i/_*/;char*d="P%"   "d\n%d\40%d"/**/
                    "\n%d\n\00wb+",b[1024],y[]="yuriyurararararayuruyuri*daijiken**akkari~n**"
          "/y*u*k/riin<ty(uyr)g,aur,arr[a1r2a82*y2*/u*r{uyu}riOcyurhiyua**rrar+*arayra*="
       "yuruyurwiyuriyurara'rariayuruyuriyuriyu>rarararayuruy9uriyu3riyurar_aBrMaPrOaWy^?"
     "*]/f]`;hvroai<dp/f*i*s/<ii(f)a{tpguat<cahfaurh(+uf)a;f}vivn+tf/g*`*w/jmaa+i`ni("/**
    */"i+k[>+b+i>++b++>l[rb";int/**/u;for(i=0;i<101;i++)y[i*2]^="~hktrvg~dmG*eoa+%squ#l2"
    ":(wn\"1l))v?wM353{/Y;lgcGp`vedllwudvOK`cct~[|ju {stkjalor(stwvne\"gt\"yogYURUYURI"[
    i]^y[i*2+1]^4;/*!*/p=(n>1&&(m[1][0]-'-'||m[1][1]  !='\0'))?fopen(m[1],y+298):stdin;
     /*y/riynrt~(^w^)],]c+h+a+r+*+*[n>)+{>f+o<r<(-m]    =<2<5<64;}-]-(m+;yry[rm*])/[*
      */q=(n<3||!(m[2][0]-'-'||m[2][1]))?stdout /*]{     }[*/:fopen(m[2],d+14);if(!p||/*
       "]<<*-]>y++>u>>+r >+u+++y>--u---r>++i+++"  <)<      ;[>-m-.>a-.-i.++n.>[(w)*/!q/**/)
return+printf("Can "  "not\x20open\40%s\40"    ""       "for\40%sing\n",m[!p?1:2],!p?/*
  o=82]5<<+(+3+1+&.(+  m  +-+1.)<)<|<|.6>4>-+(>    m-        &-1.9-2-)-|-|.28>-w-?-m.:>([28+
 */"read":"writ");for  (   a=k=u= 0;y[u];  u=2    +u){y[k++   ]=y[u];}if((a=fread(b,1,1024/*
,mY/R*Y"R*/,p/*U*/)/*         R*/ )>/*U{  */   2&& b/*Y*/[0]/*U*/=='P' &&4==/*"y*r/y)r\}
*/sscanf(b,d,&k,& A,&          i,  &r)&&        !   (k-6&&k -5)&&r==255){u=A;if(n>3){/*
]&<1<6<?<m.-+1>3> +:+ .1>3+++     .   -m-)      -;.u+=++.1<0< <; f<o<r<(.;<([m(=)/8*/
u++;i++;}fprintf   (q,    d,k,          u      >>1,i>>1,r);u  = k-5?8:4;k=3;}else
  /*]>*/{(u)=/*{   p> >u  >t>-]s               >++(.yryr*/+(    n+14>17)?8/4:8*5/
     4;}for(r=i=0  ;  ;){u*=6;u+=               (n>3?1:0);if    (y[u]&01)fputc(/*
      <g-e<t.c>h.a r  -(-).)8+<1.                >;+i.(<)<     <)+{+i.f>([180*/1*
      (r),q);if(y[u   ]&16)k=A;if                             (y[u]&2)k--;if(i/*
      ("^w^NAMORI; {   I*/==a/*"                             )*/){/**/i=a=(u)*11
       &255;if(1&&0>=     (a=                                fread(b,1,1024,p))&&
        ")]i>(w)-;} {                                       /i-f-(-m--M1-0.)<{"
         [ 8]==59/* */                                     )break;i=0;}r=b[i++]
            ;u+=(/**>>                                   *..</<<<)<[[;]**/+8&*
            (y+u))?(10-             r?4:2):(y[u]        &4)?(k?2:4):2;u=y[u/*
             49;7i\(w)/;}            y}ru\=*ri[        ,mc]o;n}trientuu ren (
             */]-(int)'`';}            fclose(          p);k= +fclose( q);
              /*] <*.na/m*o{ri{                      d;^w^;}  }^_^}}
               "   */   return  k-               -1+   /*\'   '-`*/
                     (   -/*}/   */0x01        );      {;{    }}
                        ;          /*^w^*/       ;}
```

User meaning

- The Python program `print('Hello!')` displays the text “Hello!” to the user
  - The string `'Hello!'` is arbitrary: Python just prints a sequence of six characters
  - The communicative meaning of “Hello!” as a greeting comes from English, not Python
- This is *user meaning*: a further communicative act that results from a program’s execution

A tentative diagram

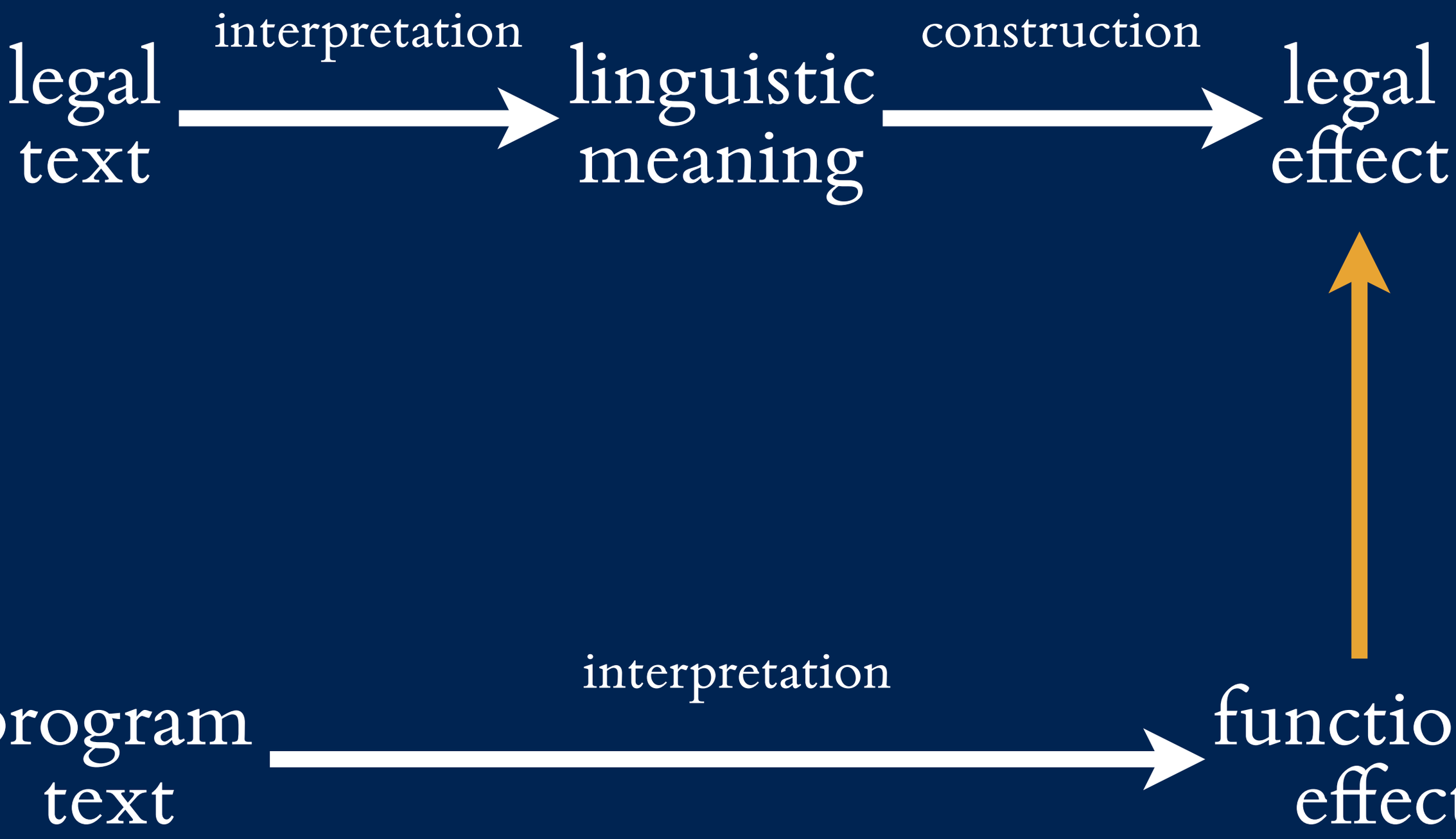


Law and software

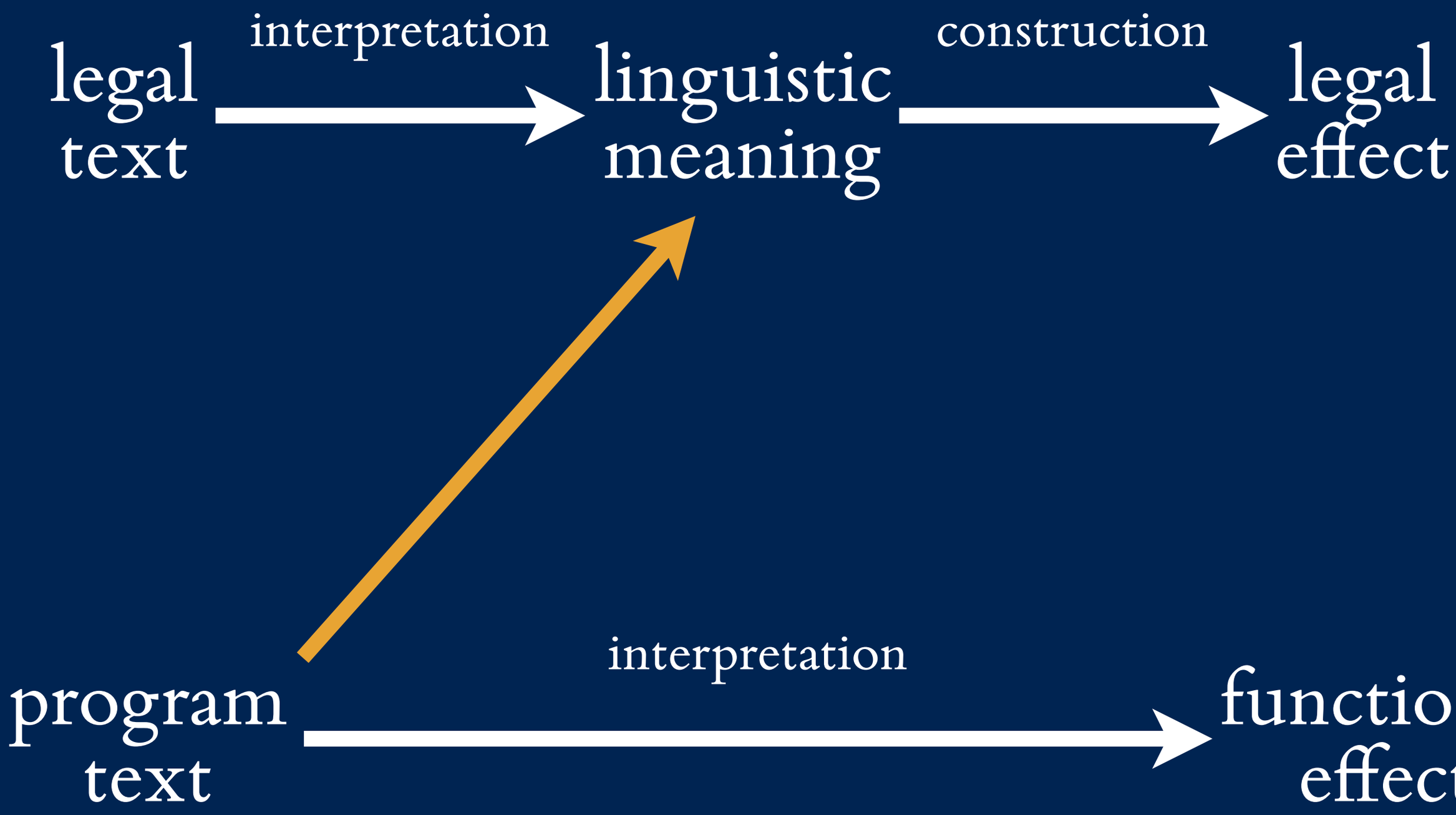
How should courts interpret software?

- Which theory of meaning is *legally* required in a given context is a question of positive law
- Which theory yields the *best results* depends on an underlying normative framework
- Computer science can tell lawyers how programmers do their jobs, but it cannot tell lawyers how to do their jobs

Executable law?



Reading software?



Back to robots.txt

- The *relevant legal question* is whether the web host has communicated the denial of permission — which requires looking at the robots.txt denial through the eyes of a reasonable web scraper, who would understand:

  (1) `Dialloq` isn't a valid keyword

  (2) The standard is written for bots to process

- (2) means that the relevant community has selected into program meaning i.e., (1) is not "corrected" to `Disallow`

Other directions

Thinking about legal interpretation

- Program meaning shows by contrast that the debate over textualism may be overblown
- The difference between textualist and other theories is far *smaller* than the difference between any of them and program meaning
- Typos do not cause statutes to crash

Ideal interpreters

- Is the ideal of a judge *another programmer* who helps the legislature test and debug its code?
- Or is the ideal of a judge *a reliable computer* who correctly executes the legislature's code?

Formalizing law

- Legal interpretation is messy because:
  1. Natural language is messy
  2. The world is messy
  3. People are messy
- Writing legal texts in software can help with (1) but not with (2), and therefore can't fully solve (3)
- Any computer system capable of interpreting natural-language texts or doing fact-finding will also be messy

On debugging

- Program analysis and testing are essential to modern software development
- They enable programmers to learn about what a program they wrote does *before* releasing it
  - (*Cf.*, the Constitution forbids the federal courts from issuing pre-release "advisory" opinions)
- Can these software development techniques be pulled back into law?

Questions?