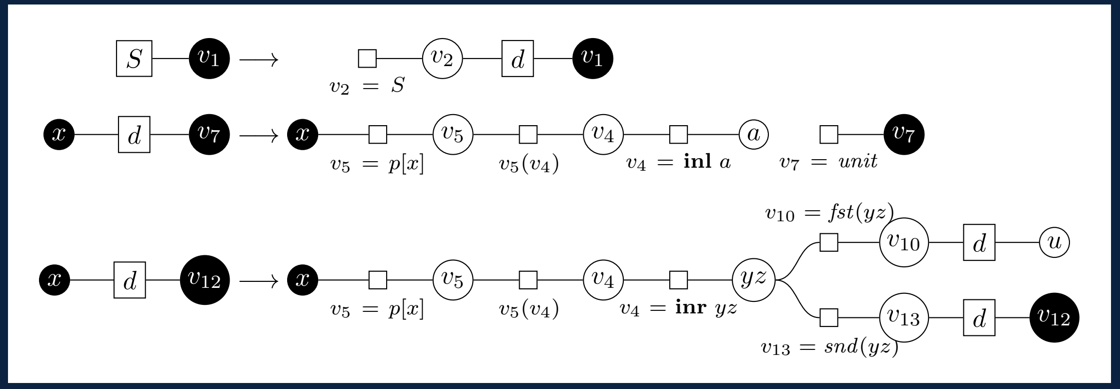
Translating Probabilistic Programs to Factor Graph Grammars

David Chiang, University of Notre Dame
Chung-chieh Shan, Indiana University

```
fun d(x) =
  case sample p[x] of
    inl a =>
      unit
  | inr yz =>
      let u = d(fst(yz)) in
      d(snd(yz));
d(S)
```

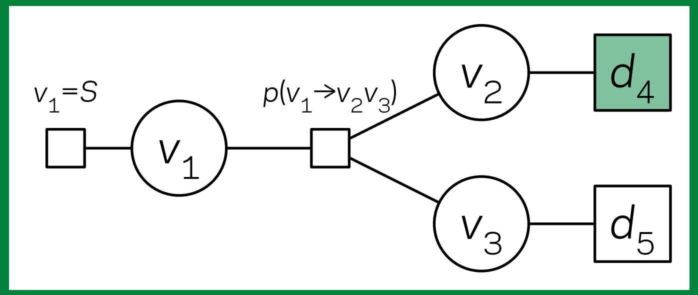
translates to



A probabilistic program with
① conditional control flow and
② unbounded recursion.

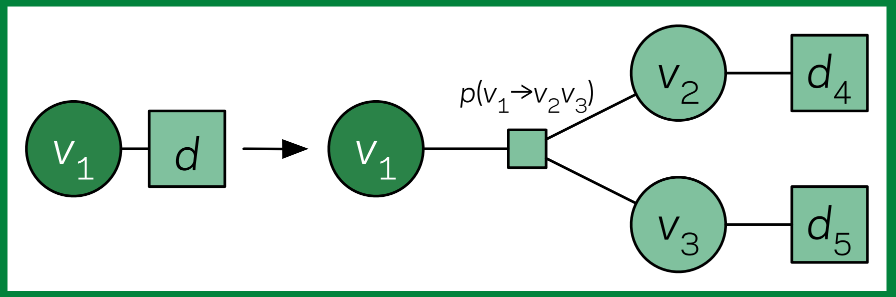
A **factor graph grammar**, which generates a set of factor graphs that together describe the same probability distribution as the program. Exact inference is possible without enumerating the (infinite) set of graphs.

Factor Graph Grammars

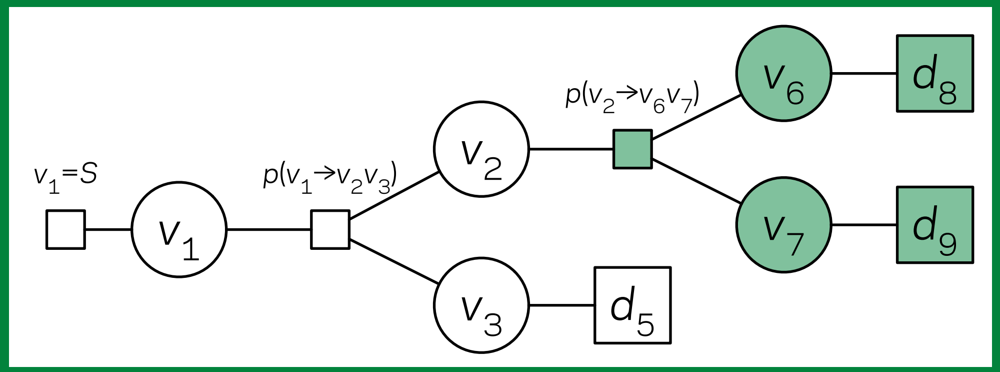


Rewrite d_4 using this rule:

rewrites to



Resulting in:



A factor graph grammar (Chiang and Riley, 2020) is a hyperedge replacement graph grammar (analogous to a context-free grammar) that generates factor graphs.

A nonterminal (*d*) rewrites to a fragment of a factor graph:

① There can be more than one possible replacement, yielding multiple alternative structures.

② The replacement can itself have nonterminals, yielding recursive structure.

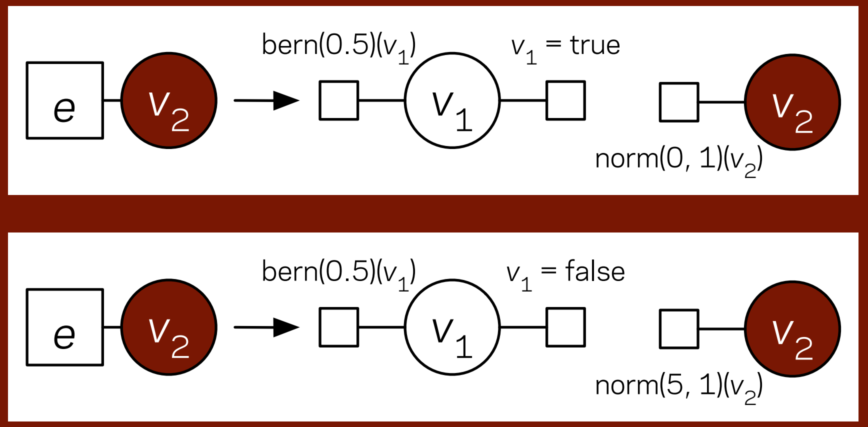
David Chiang and Darcey Riley. 2020. Factor Graph Grammars. In *Proc. NeurIPS*.

Translating Conditionals and Recursion

① Conditionals translate to two rules, one for each arm.

```
e ≡
if sample bern(0.5)
then sample norm(0, 1)
else sample norm(5, 1)
```

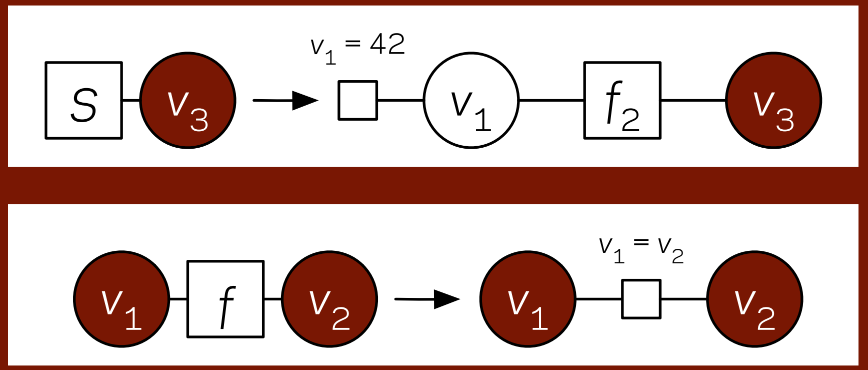
translates to



② Functions also translate to rules and can be recursive.

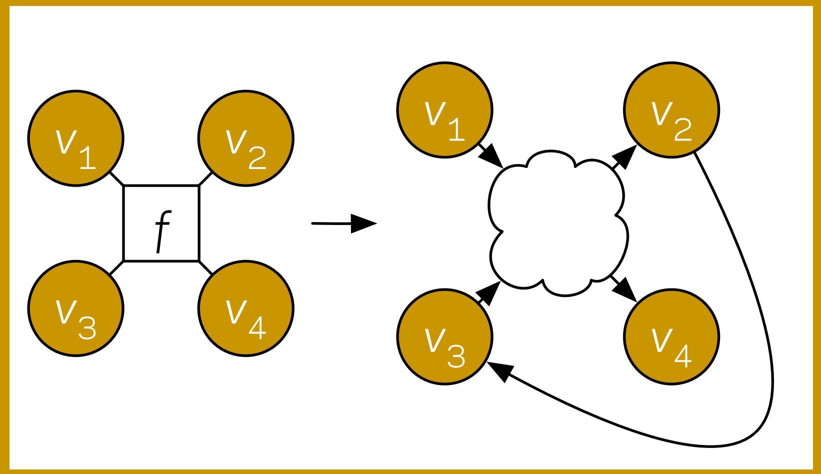
```
fun f(x) = x;
f(42)
```

translates to



Future Work

- Implement the translation: Currently only outputs LaTeX!
- Implementing FGGs is also future work. In particular, we will implement a sum-product algorithm that outputs a PyTorch computation graph.
- Extend the translation



The translation is not surjective. What kind of program would translate to a rule like the one at left?

A function with multiple outputs, and inputs can depend on outputs (but still acyclic); like a coroutine?