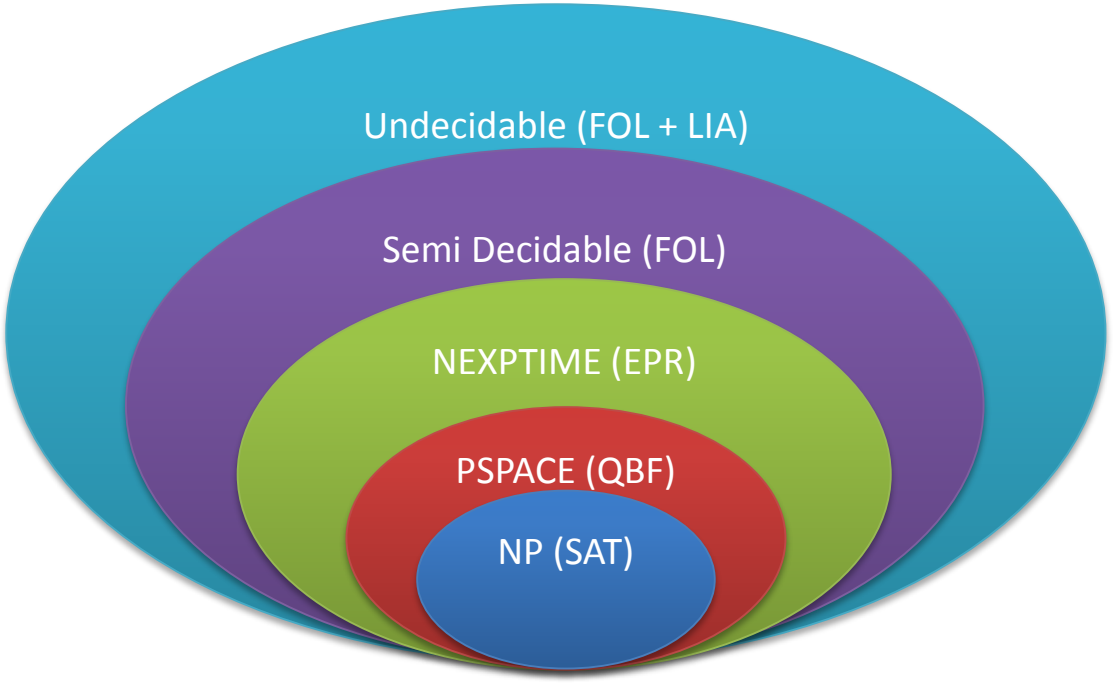
Advances in Automated Theorem Proving

Thomas Ball, MSR, representing work of many

Logic and Complexity



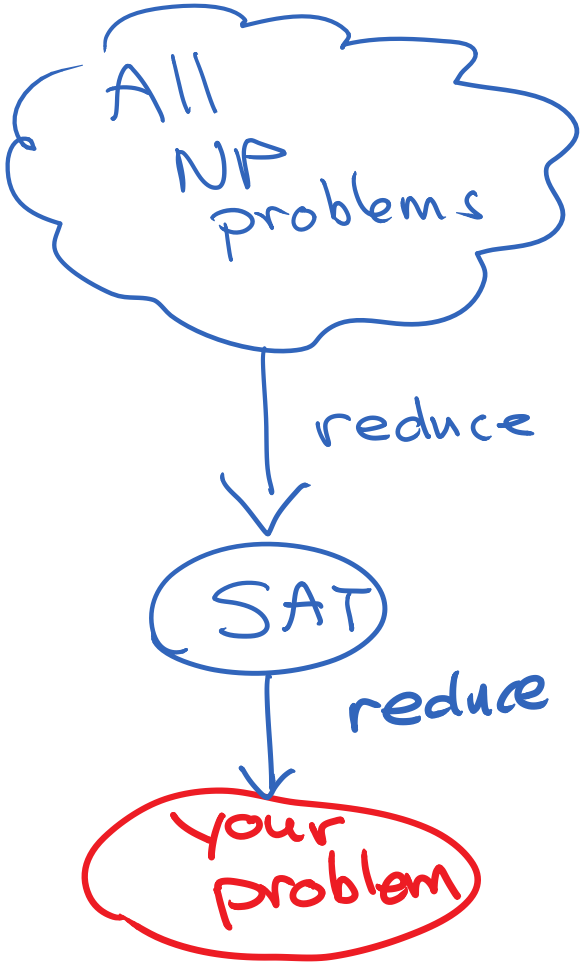
Times
Have
Changed

Then

NP-Completeness

Reduction from

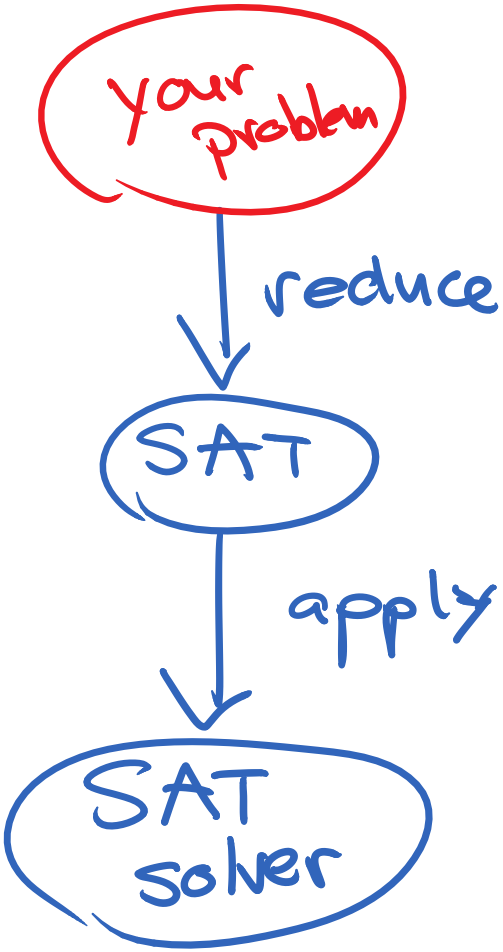
SAT



Now

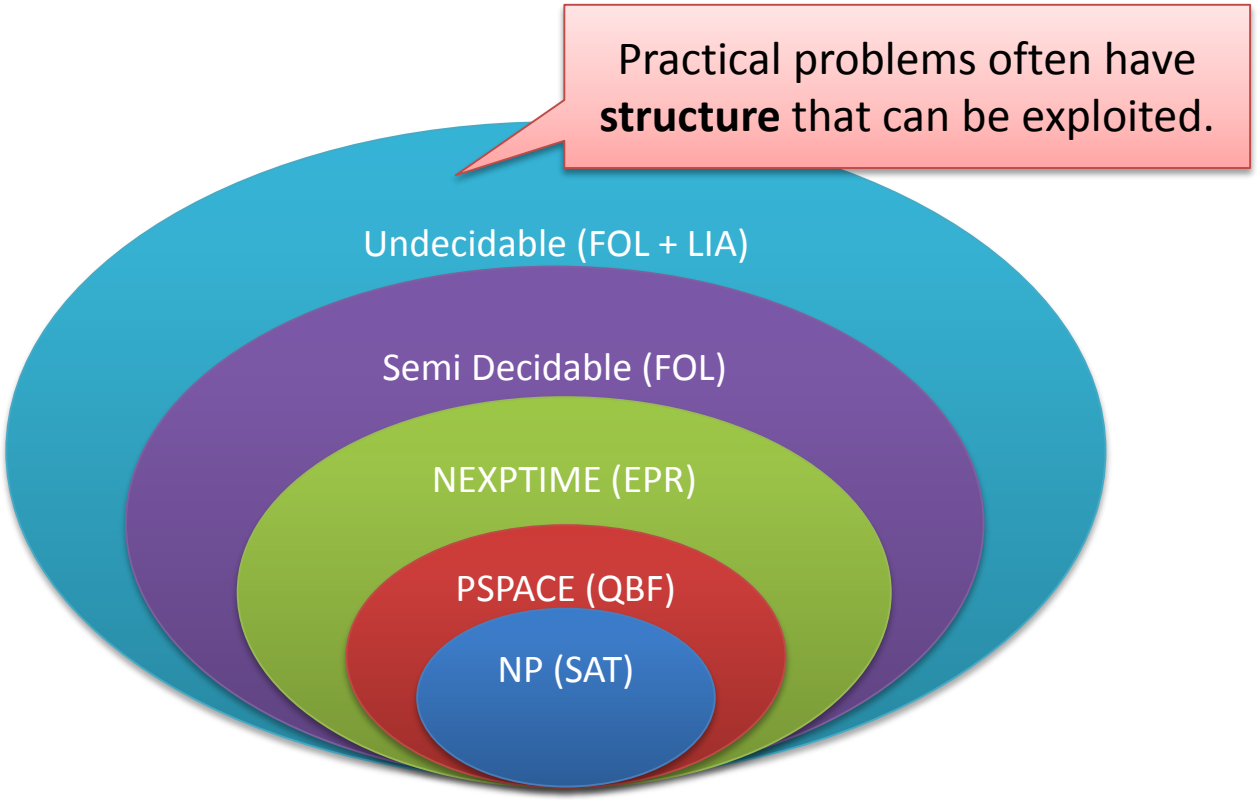
Now

Reduce your problem to SAT and solve with SAT solver

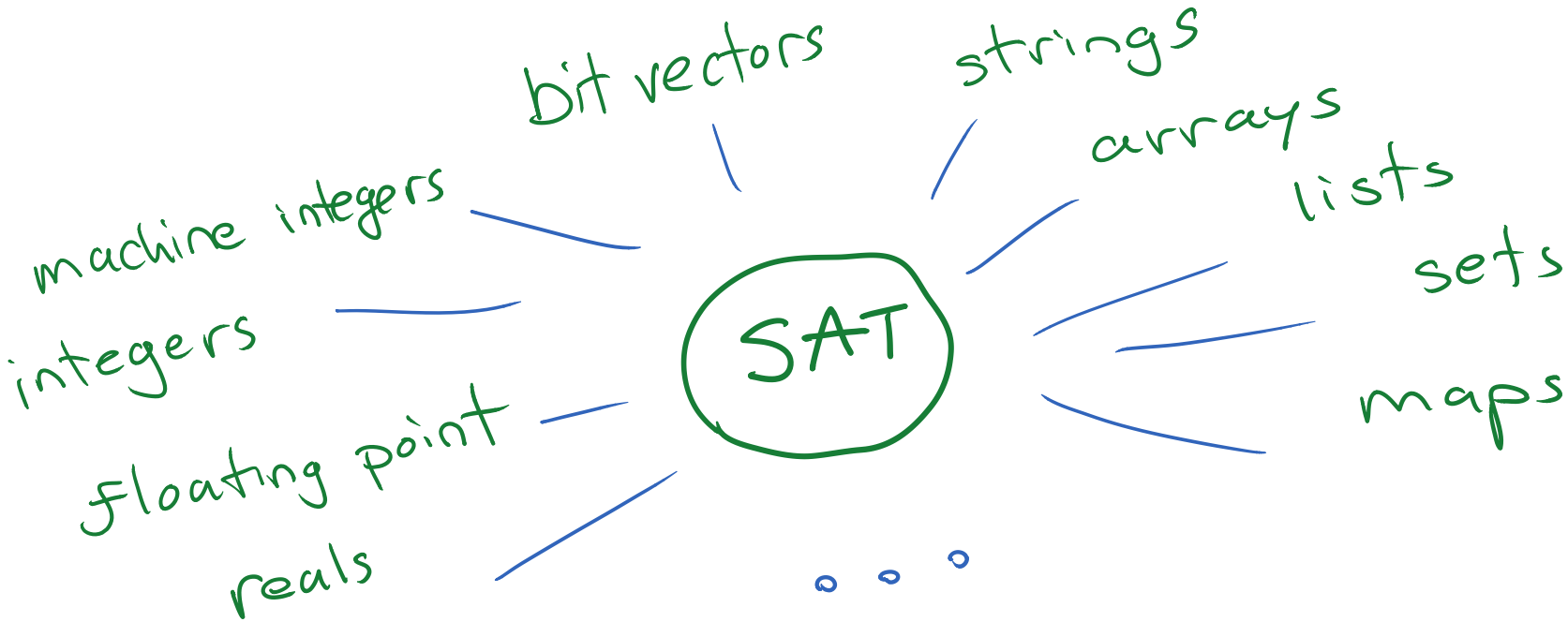


Logic and Complexity

Logic is "The Calculus of Computer Science" Zohar Manna



Satisfiability Modulo Theories (SMT)



Satisfiability

$x^2 + y^2 < 1 \; and \; xy > 0.1$ ⇒ sat, $\overbrace{x = \frac{1}{8}, y = \frac{7}{8}}^{\text{Solution/Model}}$

$x^2 + y^2 < 1 \; and \; xy > 1$ ⇒ unsat, Proof

http://research.microsoft... | Microsoft Research Mobile ...

Microsoft Research

Pex, Automated White box Testing for .NET

Pex and Moles are Visual Studio 2010 Power Tools that help Unit Testing .NET applications.

- **Pex automatically generates test suites with high code coverage.** Right from the Visual Studio code editor, Pex finds interesting input-output values of your methods, which you can save as a small test suite with high code coverage. Microsoft Pex is a Visual Studio add-in for testing .NET Framework applications.

		value
	1	null
	2	""
	3	"\0"
	4	":"

- **Moles allows to replace any .NET method with a delegate. The Fakes Framework in Visual Studio 2012 is the next generation of Moles & Stubs, and will eventually replace it.** Moles supports unit testing by providing isolation by way of detours and stubs. The Moles framework is provided with Pex, or can be installed by itself as a Microsoft Visual Studio add-in.

```
var fileName =
MFileSystem.Re
{
    Assert.IsT
    return con
```

www.pex4fun.com

Dynamic Test Generation

```
void top(char input[4])
{
   int cnt = 0;
   if (input[0] == 'b') cnt++;
   if (input[1] == 'a') cnt++;
   if (input[2] == 'd') cnt++;
   if (input[3] == '!') cnt++;
   if (cnt >= 4) crash();
}
```

input = "good"

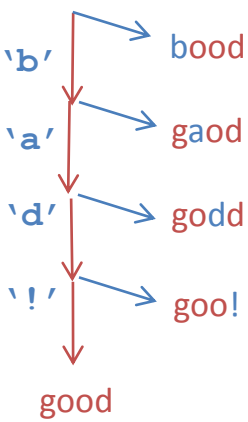
Path constraint:

I_0!='b' → I_0='b'

I_1!='a' → I_1='a'

I_2!='d' → I_2='d'

I_3!='!' → I_3='!'



Negate each constraint in path constraint

Solve new constraint → new input

Key technology: satisfiability solving

Automated Theorem Provier

http://research.microsoft.com/projects/z3/

Leonardo de Moura and Nikolaj Bjørner

SAT

Simplex

Rewriting

Superposition

Z3 is a collection of

Symbolic Reasoning Engines

Congruence Closure

Groebner Basis

∃∀ eliminatio n

Euclidean Solver

Some (MSR) Applications

- Functional verification: Dafny, VCC
- Defect detection: Corral, HAVOC, Poirot, SLAM
- Test generation: Pex, SAGE
- Design-space exploration: Formula, SpecExplorer
- New programming languages: F*

Learn about Z3 and get the source code!

- Start here
  - http://rise4fun.com/Z3/tutorial/guide
- Strategies
  - http://rise4fun.com/Z3/tutorial/strategies
- **Source code**
  - http://z3.codeplex.com/

Part 1

Revisiting Foundations in Light of SMT

Symbolic Automata and Transducers

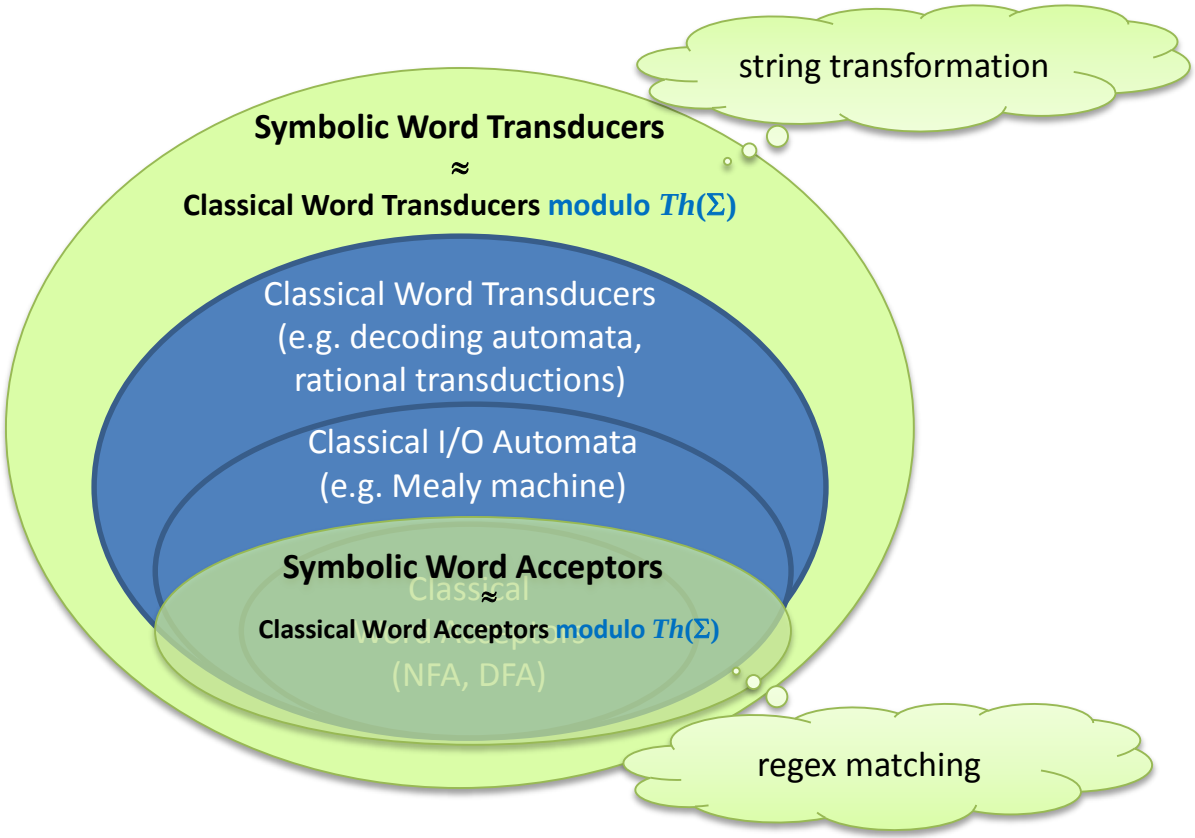
Margus Veanes, Nikolaj Bjørner

(POPL 2011)

Core Question

Can classical automata theory and algorithms be extended to work *modulo* large (infinite) alphabets $\sum$?

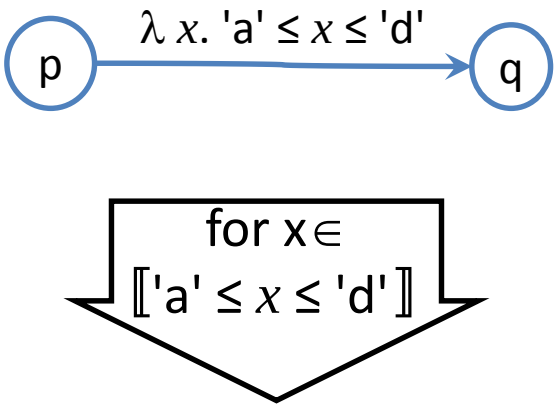
Symbolic Automata: Relativized Formal Language Theory



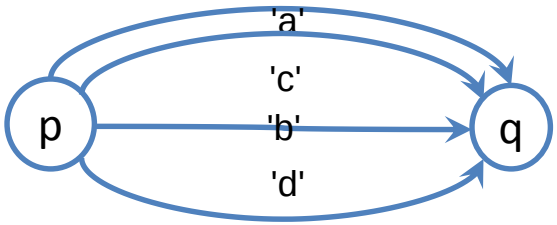
Symbolic Finite Automaton (SFA)

- Alphabet is an ***effective Boolean Algebra A***
- Labels are ***predicates over A***

one symbolic transition:

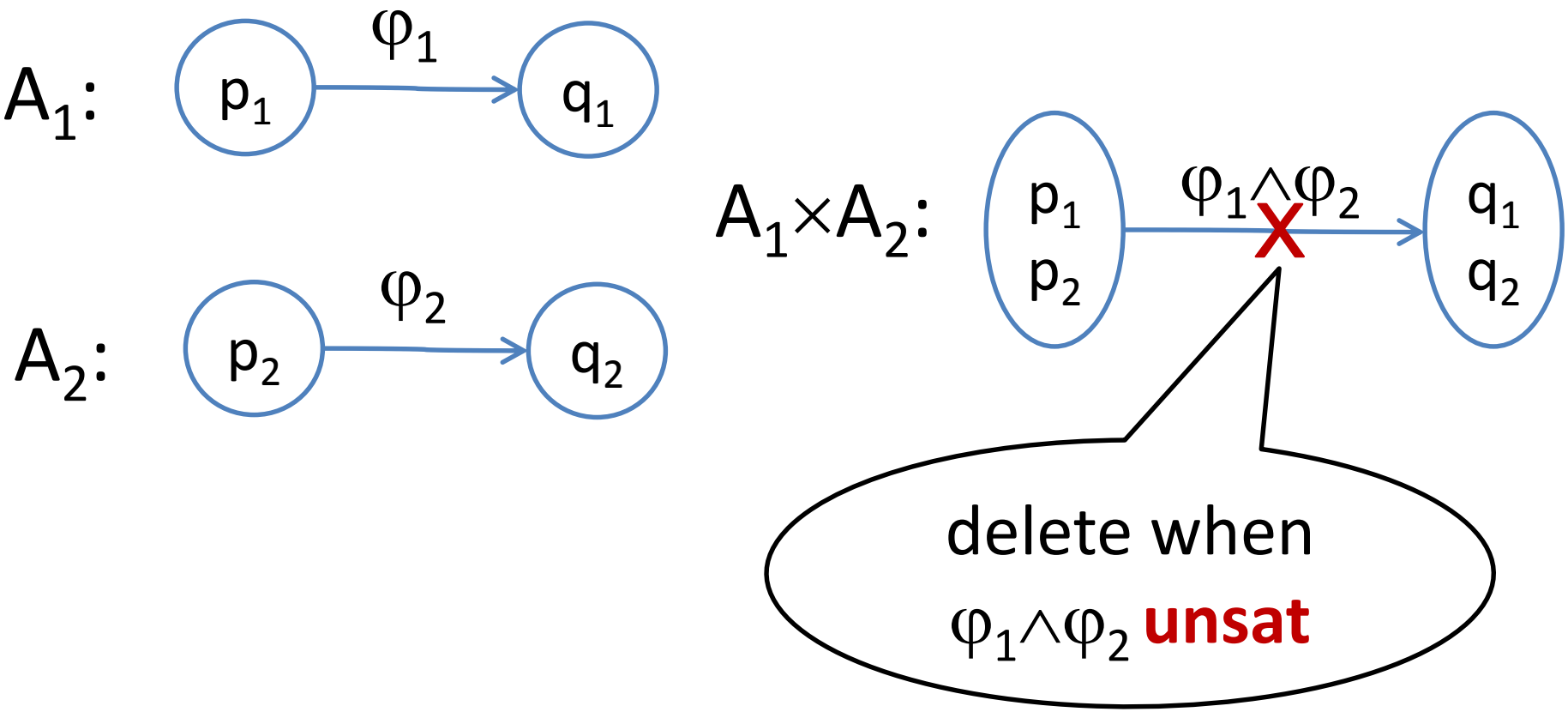


denotes **many concrete** transitions:



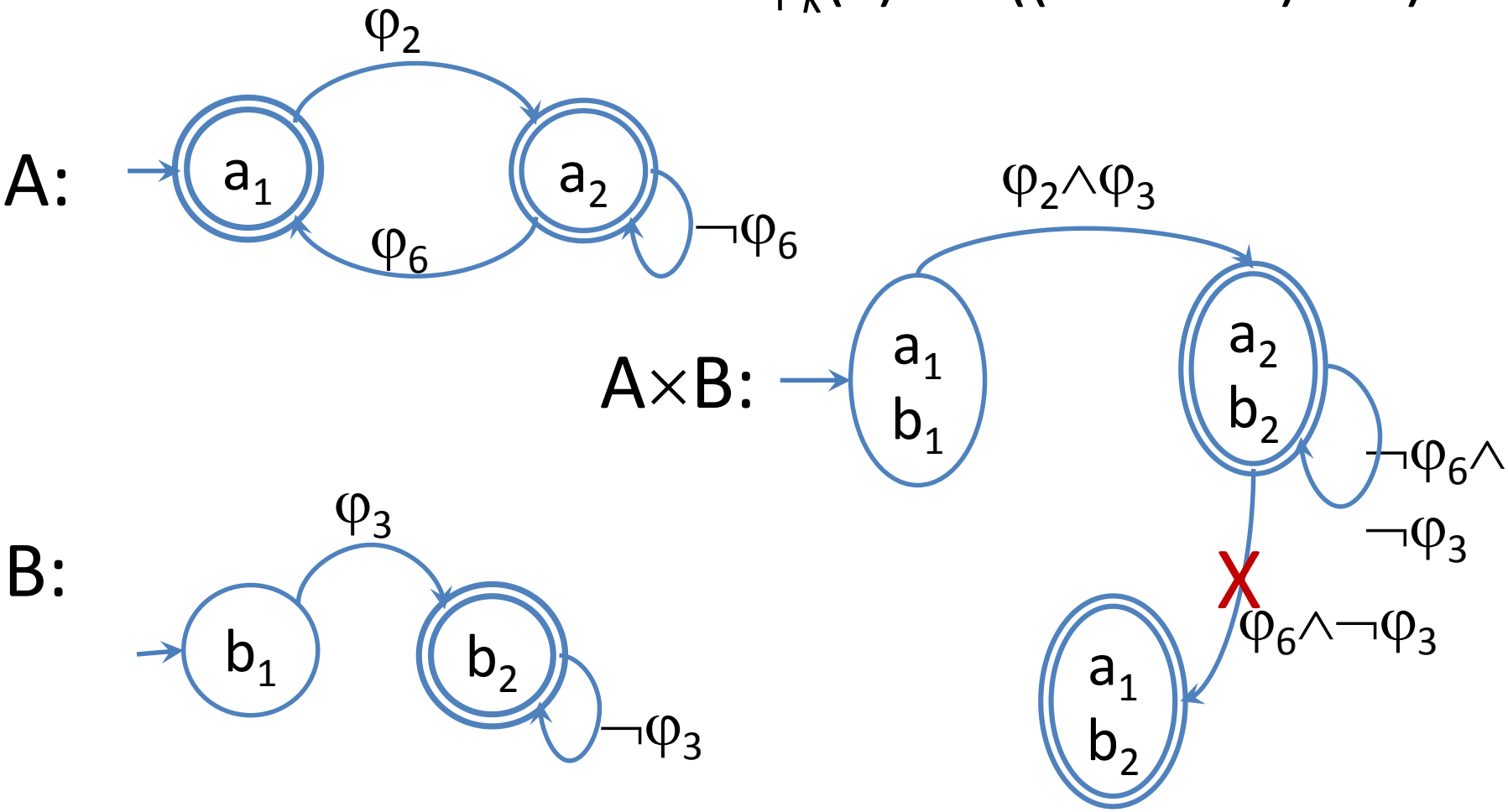
Boolean operations over SFAs

- Intersection (product of transitions)



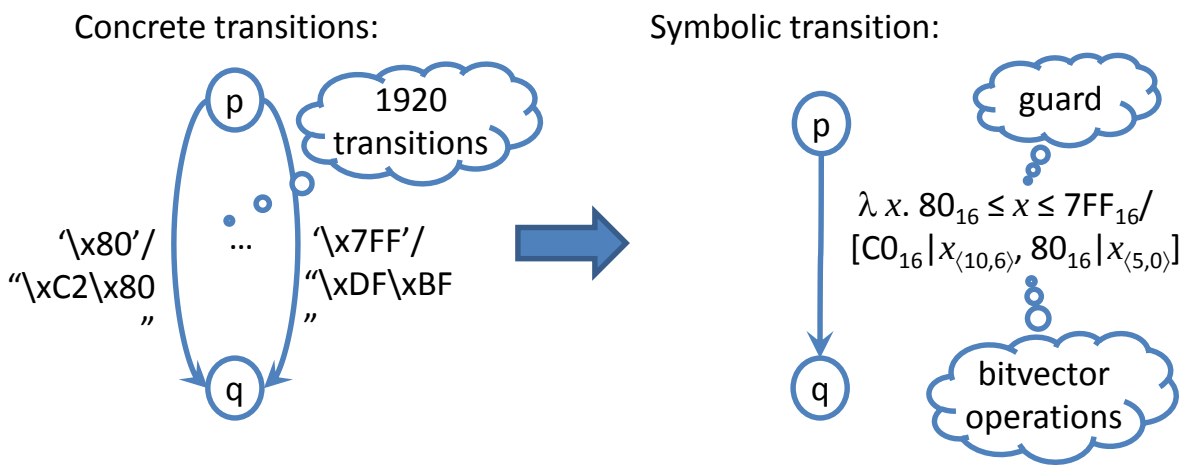
Intersection example

let $\varphi_k(x) \Leftrightarrow ((x \bmod k) = 0)$



Symbolic Finite Transducer (SFT)

- Classical transducer *modulo* a rich *label theory*
- Core Idea: represent labels with guarded transformers
  - Separation of concerns: finite graph / theory of labels



Algorithms

- **New** algorithms for SFAs and SFTs

- Extensions of classical algorithms *modulo* Th(Σ)
- Big-O complexity matches that of classical algorithms, with factor for decision procedure

Analysis

- Example 1: $\exists x(\text{utf8encode}(x) \notin R_{utf8})$?
  1. $E = SFT(\text{utf8encode})$
  2. $A = Complement(SFA(R_{utf8}))$
  3. $B = \lambda\, x.\, A(E(x))$
  4. $B \neq \emptyset$?

Does there exist an input x that causes a bad output ?

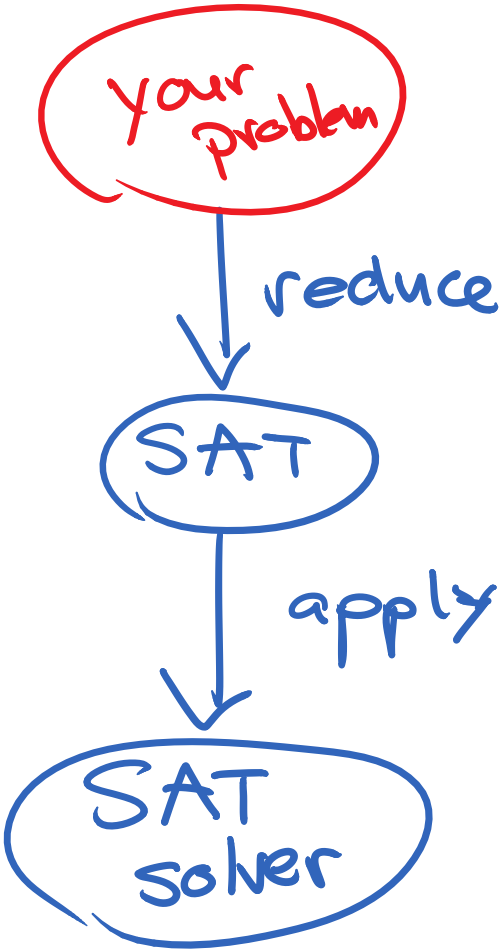
- Example 2: $\lambda\, x.\text{utf8decode}(\text{utf8encode}(x)) \equiv Id$?

For More Information

- Bek
  http://rise4fun.com/Bek/tutorial
- Bex
  http://rise4fun.com/Bex/tutorial
- Rex
  http://rise4fun.com/rex/
- Fast
  http://rise4fun.com/Fast/tutorial

A Recipe for Success?

Reduce your problem to SAT and solve with SAT solver



Part 2. The Secret Sauce

Harnessing the power of symbolic abstractions in your domain

Idea:

Represent exponentially many concrete objects efficiently (with useful operations)

Syntactic String Transformations (from Examples)

Flash Fill feature in Excel 2013

Demo!

Reference:
Automating String Processing in Spreadsheets using Input-Output Examples, POPL 2011, Gulwani

Syntactic String Transformations: Language

Trace Expr e := Concatenate(f_1, …, f_n)

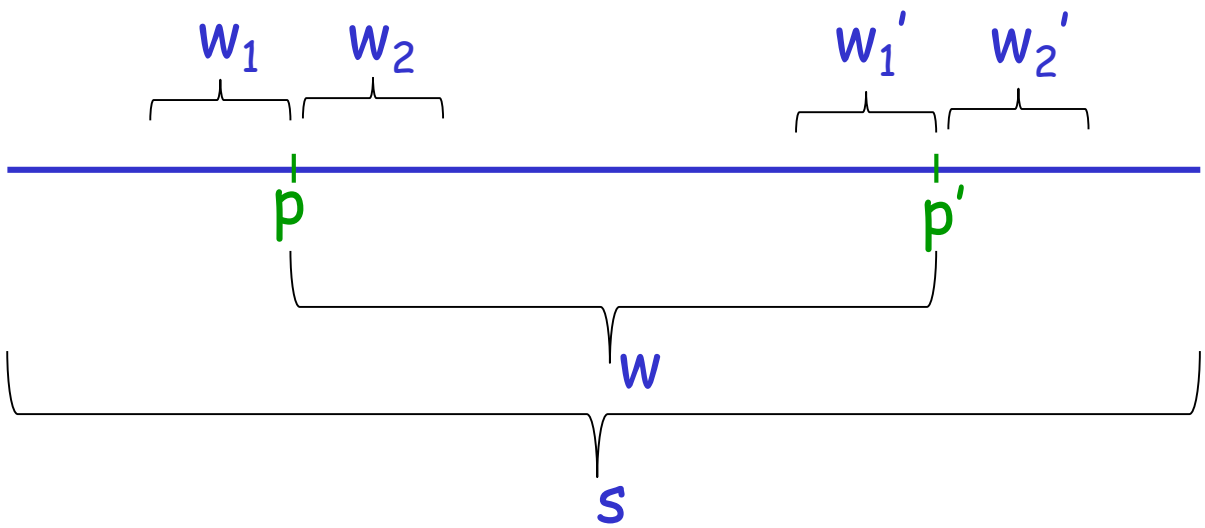
Base Expr f := s
| SubStr(v_i, p_1, p_2)

Position Expr p := k
| Pos(r_1, r_2, k)

Substring Operator

Let w = SubString(s, p, p')

where p = Pos(r_1, r_2, k) and p' = Pos(r_1', r_2', k')



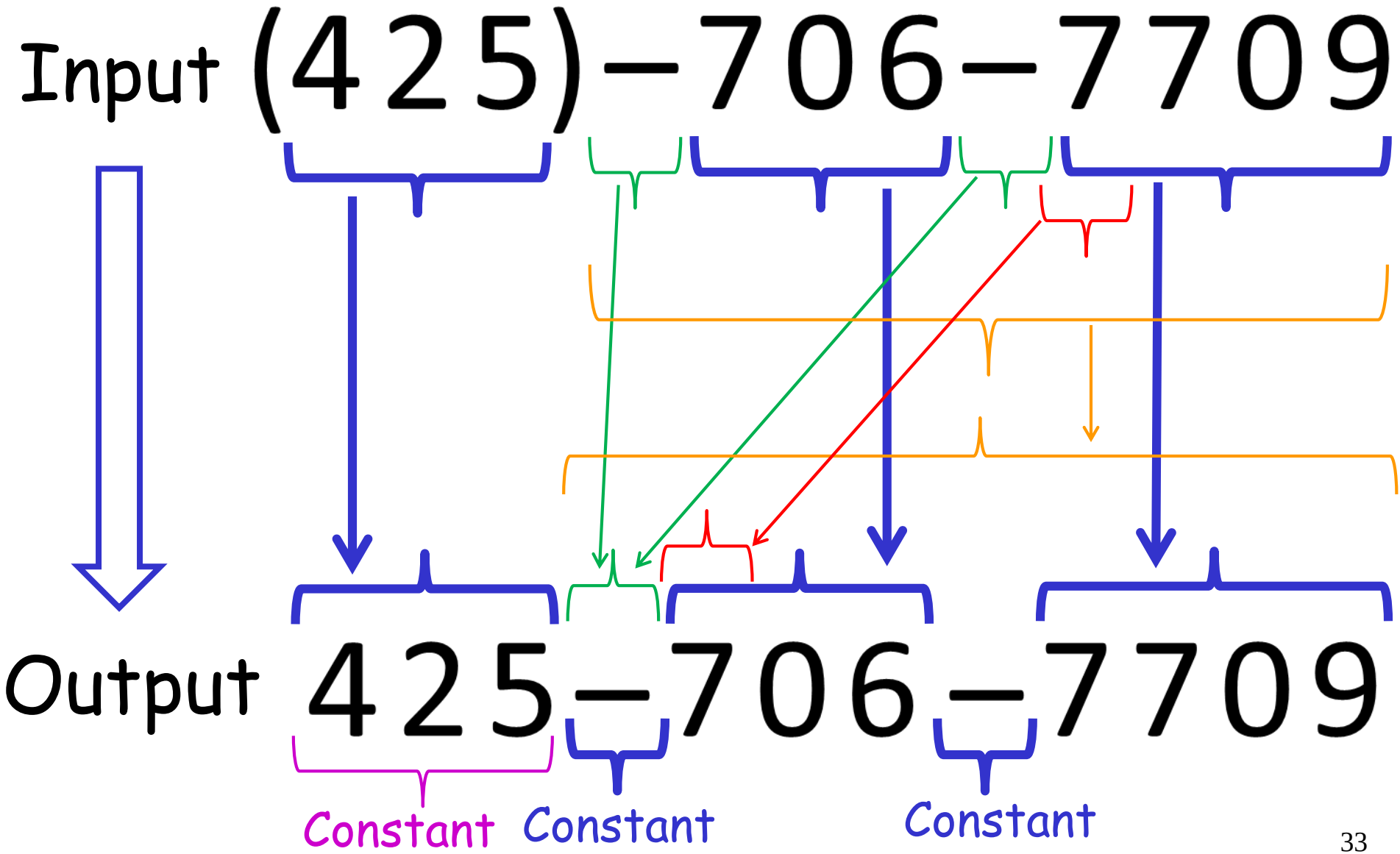
r_1 matches w_1
r_2 matches w_2

r_1' matches w_1'
r_2' matches w_2'

Two special cases:

- $r_1 = r_2' = \epsilon$: This describes the substring
- $r_2 = r_1' = \epsilon$: This describes boundaries around the substring

Too many choices for a Trace Expression



Synthesizing Trace Expressions

Number of all possible trace expressions is exponential in size of output string.

To represent/learn all trace expressions, it suffices to represent/learn all base expressions for each substring of the output.

- # of substrings is just quadratic in size of output string!
- A DAG based data-structure represents all trace expressions transforming input to output, and supports efficient intersection operation

Too many choices for a SubStr Expression

Various ways to extract "706" from "425-706-7709":

- Chars after 1st hyphen and before 2nd hyphen.
  Substr(v_1, Pos(HyphenTok,ϵ,1), Pos(ϵ,HyphenTok,2))
- Chars from 2nd number and up to 2nd number.
  Substr(v_1, Pos(ϵ,NumTok,2), Pos(NumTok,ϵ,2))
- Chars from 2nd number and before 2nd hyphen.
  Substr(v_1, Pos(ϵ,NumTok,2), Pos(ϵ,HyphenTok,2))
- Chars from 1st hyphen and up to 2nd number.
  Substr(v_1, Pos(HyphenTok,ϵ,1), Pos(ϵ,HyphenTok,2))

⋮

Synthesizing SubStr Expressions

The number of SubStr(v,p_1,p_2) expressions that can extract a given substring w from v can be large!

To represent/learn all SubStr expressions, we can independently represent/learn all choices for each of the two index expressions.

This allows for representing and computing $O(n_1*n_2)$ choices for SubStr using size/time $O(n_1+n_2)$.

Ranking

General Principles

- Prefer shorter programs, shorter string expression, regular expressions.
- Prefer programs with less number of constants.

Conclusions

- Logic as our calculus
- Automation
- Logical thinking