Assignment Update

- We have our first pull requests with passing/failing programs
  - thanks to Jakub Z., Daniel D., and Thomas P.
- Tutorial (code review) on Wednesday at 13:30-14:15
  - we will start on an implementation of SymbolicDictionary
- I soon will write a paper on dynamic symbolic execution, including your contributions
  - **Acknowledgements** for people who submit passing/failing tests
  - **Co-authorship** for people who contribute significant idea or code contribution
    - New symbolic data type
    - Fix of failing tests (improve coverage)
    - …

An interesting test case: saga of control/data

```
A = [0, 1]

def arrayindex(a):
  if A[a]:
    return A[a]
  else:
    return "OTHER"
```

Since A is finite and iterable, we can invert it!

```
A = [0, 1]

def arrayindex(a):
  if A[a]:
    return A[a]
  else:
    return "OTHER"
```

```
A = [0, 1]

def arrayindex2(a):

 b  = false
 for x in A if A[x]:
   b = b or (a==x)

if b:
   return A[a]
 else:
   return "OTHER"
```

The need for applying multiple theories of integer arithmetic

- In Python, (a+1 < a) never evaluates to true
- Using BitVectors it takes too much time to “prove” this fact
- First apply Linear Integer Arithmetic (LIA) + Equality with Uninterpreted Functions (EUF)
  - If UNSAT, return UNSAT
  - If SAT, check result (may be incorrect because of EUF)
  - If UNKNOWN, apply BitVector

Lecture 4

Strings, Regular Expressions and Symbolic Automata

via

Z3's algebraic datatypes and quantifiers

Thanks to Margus Veanes, Nikolaj Bjorner

Solving *regular membership constraints*

- Example:
  - `x` is a valid email address:
    `^[A-Za-z0-9]+@(([A-Za-z0-9\-])+\.)+([A-Za-z\-])+$`
  - `x` starts and ends with letter m
    `^m.*m$`

- Solution to both constraints: `x = m@v.com`

- When using an SMT solver, can be combined with other (e.g. arithmetical) constraints
  - `Length(x) < y + z`

Supporting regex constraints in DSE

```
if re.match(x, "^m.*m$"):
    …
else:
    …
```

Find a value for parameter x to reach this branch

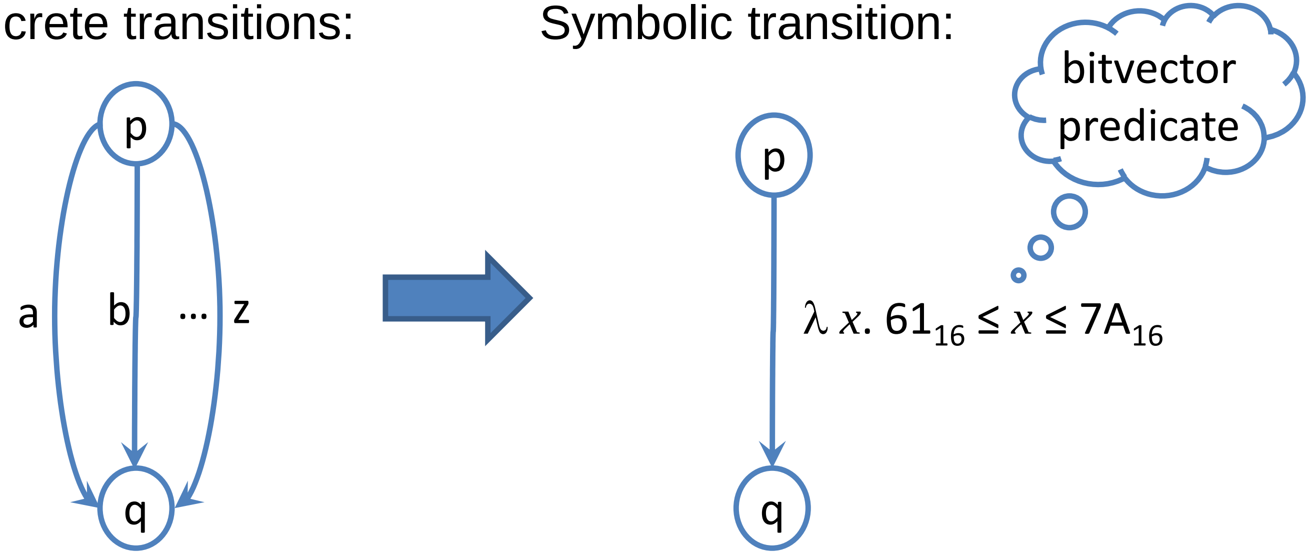
The plan…

- A *regex* r is translated into a *symbolic finite acceptor (SFA)* A_r
- A_r is given to a *SMT solver (Z3)*
  - The solver may support other constraints

Symbolic Finite Acceptor (SFA)

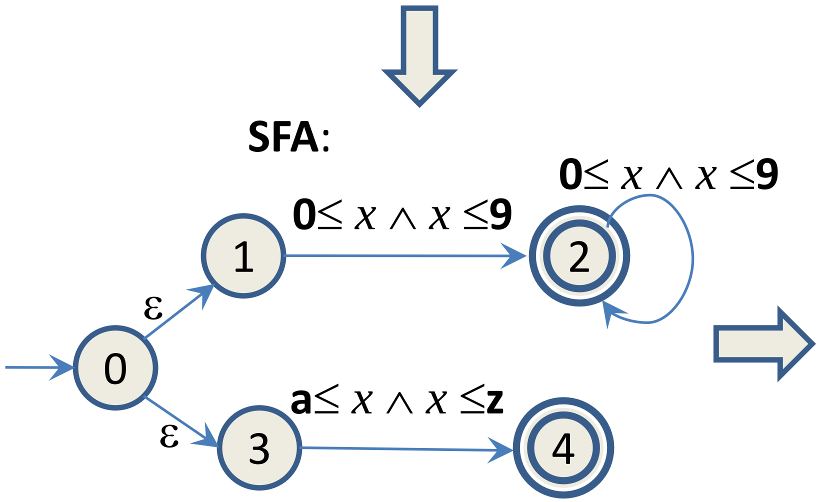
- Classical acceptor *modulo* a rich alphabet
  - Alphabet is an *effective Boolean Algebra*
- Core idea: represent labels with predicates
  - Separation of concerns: finite graph / algebra of labels

Concrete transitions: Symbolic transition:



SFA axioms ($Th(A)$)

Regex: **\d+|[a-z]**



Theory

(y:List<Σ>)

$Acc_0(y) \Leftrightarrow Acc_1(y) \vee Acc_3(y)$

$Acc_1(y) \Leftrightarrow y \neq [] \wedge \text{‘0’} \le first(y) \le \text{’9’} \wedge Acc_2(rest(y))$

$Acc_2(y) \Leftrightarrow y = [] \vee$

$\quad (y \neq [] \wedge \text{‘0’} \le first(y) \le \text{’9’} \wedge Acc_2(rest(y)))$

$Acc_3(y) \Leftrightarrow y \neq [] \wedge \text{‘a’} \le first(y) \le \text{’z’} \wedge Acc_4(rest(y))$

$Acc_4(y) \Leftrightarrow y = []$

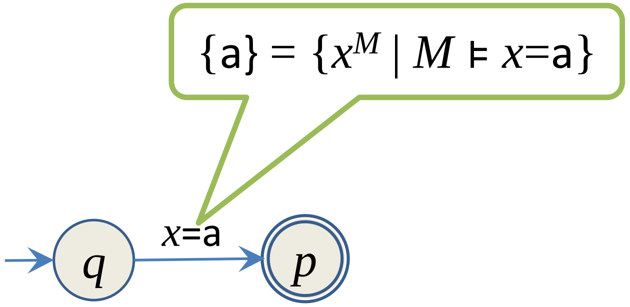
Note: a move $(p, \varphi[x], q)$ encodes the *set* of transitions

$$\{(p, x^M, q) \mid M \vDash \varphi[x]\}$$

Conditional correctness of $Th(A)$

- **Theorem***: Let A be an FSA without ε-loops. $Th(A) \wedge Acc^A(s,k)$ is sat. $\Leftrightarrow$ $s \in L(A)$ and $len(s)=k$.
  - The theorem fails if ε-loops are allowed.
  - During regex to SFA construction, ε-loops are eliminated, but some ε-moves may remain. Full ε-elimination may increase the number of moves considerably and slow down the analysis

Step-by-step example ($Th(A)$ construction)

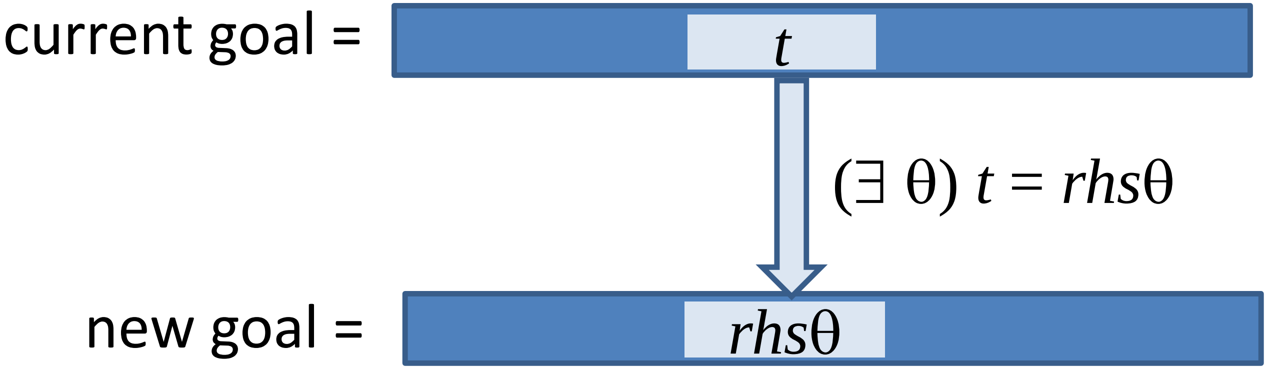


- Given regex r: “a”, i.e. $L(r)$={a}
- Construct automaton A
- Define $Th(A)$:
  - $\forall s\, Acc_q(s,0) \Leftrightarrow false$ (q is not final)
  - $\forall s\, n\, Acc_q(s,succ(n)) \Leftrightarrow hd(s)=\mathrm{a} \wedge Acc_p(tl(s),n)$
  - $\forall s\, Acc_p(s,0) \Leftrightarrow s=nil$ (p is final)
  - $\forall s\, n\, Acc_p(s,succ(n)) \Leftrightarrow false$ (p has no outgoing moves)

In general, axioms may also be nonequational and are *triggered* by associated *patterns*

E-matching in Z3

- Equational axioms have the form
  $\forall\, \boldsymbol{x}\ (lhs[\boldsymbol{x}] = rhs[\boldsymbol{x}])$
  (Note that '$=$' is same as '$\Leftrightarrow$' when $lhs, rhs{:}\mathbb{B}$)
- There is a *current goal* that is a *quantifier free ground* formula, axioms are used to *rewrite* the goal during model generation by matching axioms (from left to right):



Step-by-step example (*solving*)

- Assuming $Th(A_r)$ as defined earlier for r="a"
- Declare $w{:}\mathbb{L}\langle\mathbb{C}\rangle$ as an *uninterpreted constant*
- Consider the goal $Acc_q(w, succ(0))$
- E-matching:

$Acc_q(w, succ(0))$

$\Downarrow$

$(\exists\ \theta=\{s \mapsto w, n \mapsto 0\})\ Acc_q(w, succ(0)) = Acc_q(s, succ(n))\theta$
Thus: replace $Acc_q(w, succ(0))$ with $hd(s)=\mathrm{a} \wedge Acc_p(tl(s),n)\theta$

$hd(w)=\mathrm{a} \wedge Acc_p(tl(w),0)$

$\Downarrow$

$(\exists\ \theta=\{s \mapsto tl(w)\})\ Acc_p(tl(w),0) = Acc_p(s,0)\theta$
Thus: replace $Acc_p(tl(w),0)$ with $(s=nil)\theta$

$hd(w)=\mathrm{a} \wedge tl(w)=nil$

- Now $hd(w)=\mathrm{a} \wedge tl(w)=nil$ has a model M using the built-in list theory, namely $w^M = cons(\mathrm{a},nil)$

Algorithms on SFAs

- There are straightforward generalizations of classical algorithms of (N)FAs to SFAs, such as:
  1. Epsilon elimination
  2. Determinization
  3. Minimization
  4. **Product construction**

Product construction of SFAs

- Given A and B construct C, $L(C) = L(A) \cap L(B)$

(i) Initially $S = (\langle q_{0A}, q_{0B}\rangle)$, $V = \{\langle q_{0A}, q_{0B}\rangle\}$, $T = \emptyset$.

(ii) If S is empty go to (iv) else pop $\langle q_1, q_2\rangle$ from S.

(iii) Iterate for each $t_1 \in \Delta_A(q_1)$ and $t_2 \in \Delta_B(q_2)$, let $\varphi = \mathit{Cond}(t_1) \wedge \mathit{Cond}(t_2)$, let $p_1 = \mathit{Target}(t_1)$, and let $p_2 = \mathit{Target}(t_2)$. If φ *is satisfiable* then

- add $(\langle q_1, q_2\rangle, \varphi, \langle p_1, p_2\rangle)$ to T;
- if $\langle p_1, p_2\rangle$ is not in V then add $\langle p_1, p_2\rangle$ to V and push $\langle p_1, p_2\rangle$ to S.

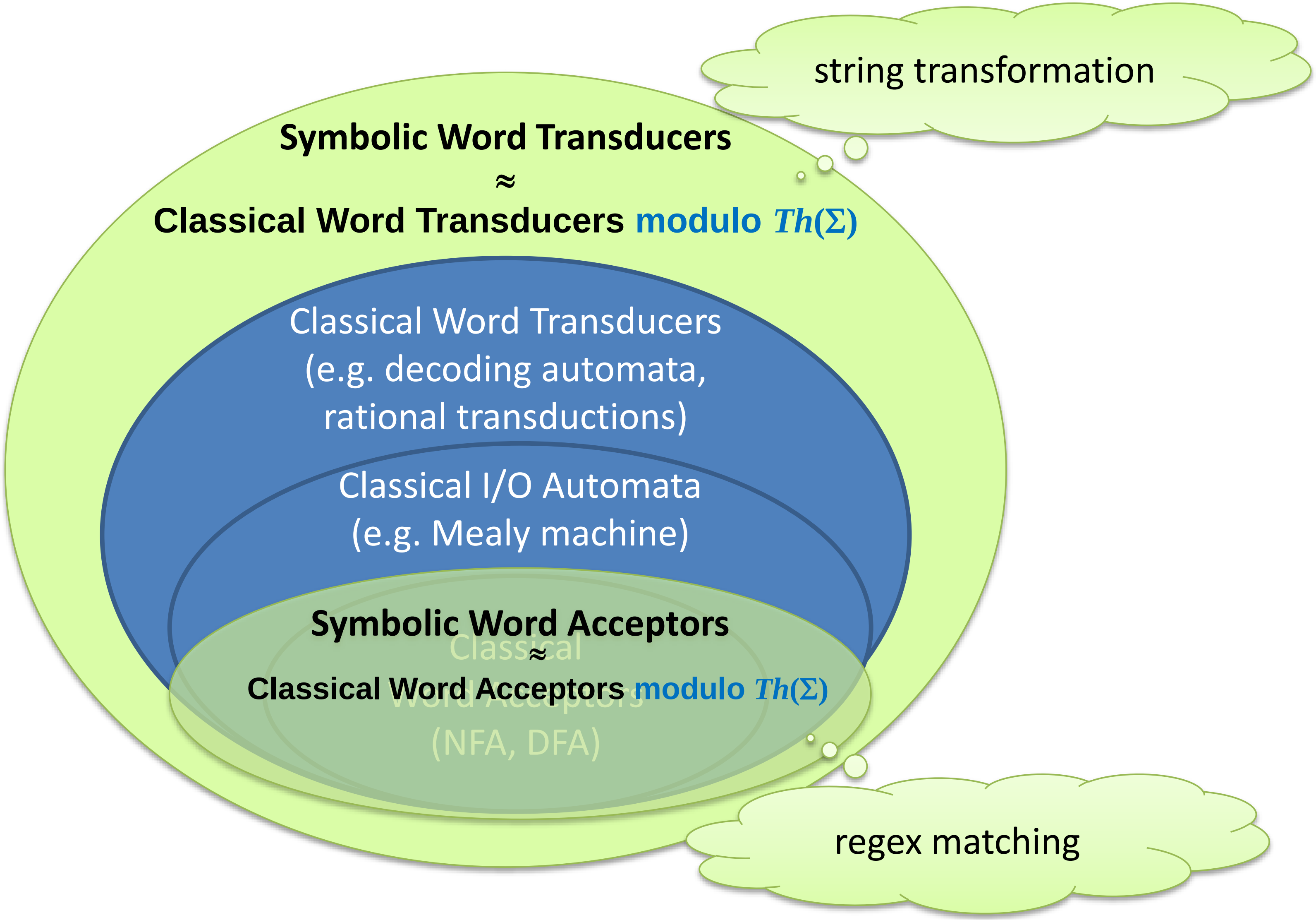
Proceed to (ii).

(iv) Let $C = (\langle q_{0A}, q_{0B}\rangle, V, \{q \in V \mid q \in F_A \times F_B\}, T)$.

(v) Eliminate *dead states* from C (states from which no final state is reachable).

Using constraint solver

Relativized Formal Language Theory



Core Question

- Can classical automata theory and algorithms be extended to work *modulo* large (infinite) alphabets Σ ?

  Not obvious: e.g.
  NFA *determinization* is $\mathcal{O}(|\Sigma|2^n)$, DFA *minimization* is $\mathcal{O}(|\Sigma|n\log n)$,

why?

- Analysis of: string **acceptors**
  - regexes (Σ is Unicode)

how?

- **Symbolic Finite Acceptors**
  (application) *ICST 2010*
  (theory) *LPAR 2010*
  (evaluation) *VMCAI 2011*

why?

- Analysis of: string **transformers**
  - sanitizers, encoders, decoders

how?

- **Symbolic Finite Transducers**
  (evaluation) *USENIX Security 2011*
  (theory) *POPL 2012, VMCAI 2013*
  (tool) *TACAS 2012*

Assignment

- Add basic support for strings into PyExZ3

- Using the theory of lists, how much can we do?
  - Constants
  - Concatenation (+)
  - Substring matching (and extraction?)

Summary: a white-box approach to automated test generation

- Dynamic Symbolic Execution
  - Execute a program P on concrete input I
  - Collect symbolic constraints characterizing P(I)
  - Negate constraints to find new input

- Satisfiability Modulo Theories Solvers (like Z3)
  - SAT solver for propositional logic
  - SMT = SAT + Theories
  - bitvectors, arrays, algebraic data types, quantifiers

Let’s Party!

Code review on Wednesday at 13:30

Get latest code from me or github…

After the school, contribute/discuss via
http://www.github.com/thomasjball/PyExZ3/

Email: tball@microsoft.com