Up-To Techniques for Weighted Systems

Barbara König

Universität Duisburg-Essen

Joint work with Filippo Bonchi (Università di Pisa)
& Sebastian Küpper (FernUniversität Hagen)

TACAS 2017

Overview

Motivation

Weighted Automata

Weighted automata are the quantitative variant of (non-deterministic) finite automata.

Instead of checking whether a work is in the language $(0, 1)$, they assign to every word a weight, i.e. an element from a given semiring.

Applications, for instance in natural language processing.

Motivation

Our aim

Efficient techniques for solving problems on weighted automata:

- Language equivalence
  Are the languages accepted by two given automata equal?
- Language inclusion
  Given two automata, does the first automaton assign to each word weights smaller (or equal) than the weights of the second automaton?
- Threshold/Universality
  Is the weight of each word above some given threshold T?

Our approach

Use so-called up-to techniques (known from process algebra).
"Up-to" is used in the sense of "modulo".

Weighted Automaton over a Semiring

We consider weighted automata over arbitrary semirings:

Semiring

Tuple $(\mathbb{S}, \oplus, \otimes, 0, 1)$ where

- $\mathbb{S}$ is the carrier set,
- $(\mathbb{S}, \oplus, 0)$ is a commutative monoid,
- $(\mathbb{S}, \otimes, 1)$ is a (commutative) monoid,
- $\otimes$ distributes over $\oplus$ and 0 is an annihilator for $\otimes$.

Examples

- Arithmetic semiring (reals): $(\mathbb{R}, +, \cdot, 0, 1)$
- Tropical semiring: $(\mathbb{N}_0 \cup \{\infty\}, \min, +, \infty, 0)$
- Distributive lattices: $(\mathbb{L}, \sqcup, \sqcap, \bot, \top)$

Weighted Automaton over a Semiring

Vectors over a Semiring

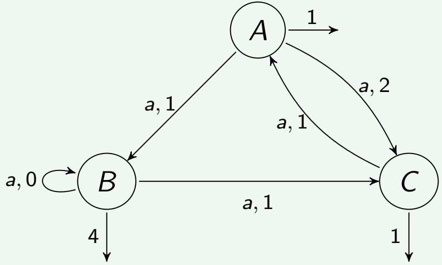
We consider vectors of the form $v\colon X \to \mathbb{S}$, where X is a (finite) set.

Weighted Automaton

Given an alphabet Σ, a weighted automaton is a triple (X, o, t) where

- X is a (finite) set of states
- $o\colon X \to \mathbb{S}$ is the output function.
- $T_a\colon X \times X \to \mathbb{S}$, $a \in \Sigma$ are transition matrices

Weighted Automaton over a Semiring



tropical semiring $\quad \Sigma = \{a\} \quad T_a = \begin{pmatrix} \infty & 1 & 2 \\ \infty & 0 & 1 \\ 1 & \infty & \infty \end{pmatrix}$

$o = \begin{pmatrix} 1 \\ 4 \\ 1 \end{pmatrix} \quad$ Initial (column) vector $i = \begin{pmatrix} 0 & \infty & \infty \end{pmatrix}$

Weighted Automaton over a Semiring

Weight of a Word

For a given initial vector i, the weight of a word $w = a_1 \dots a_n$ is

$$[\![i]\!](w) = i \cdot T_{a_1} \cdot \cdots \cdot T_{a_n} \cdot o$$

where $\cdot$ denotes matrix multiplication with $\oplus$ and $\otimes$.

Intuitively:

- for each path corresponding to w, multiply ($\otimes$) the weights
- *and* add ($\oplus$) the weights for all paths.

$$[\![i]\!](aa) = \min\{\underbrace{0+1+1+1}_{A\to B\to C}, \underbrace{0+2+1+1}_{A\to C\to A}, \underbrace{0+1+0+4}_{A\to B\to B}, \underbrace{\infty+\dots}_{B\to\dots}, \underbrace{\infty+\dots}_{C\to\dots}\} = 3$$

Problems for Weighted Automata

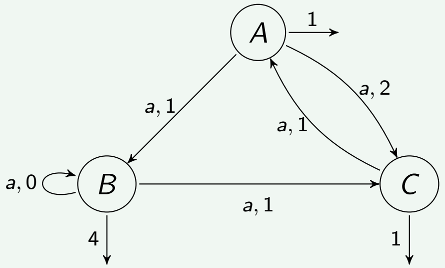
Language of a Weighted Automaton

For a given initial vector i, the mapping $[\![i]\!]\colon \Sigma^* \to \mathbb{S}$ is called the *language* of i.

Problems

- Language equivalence
  Given two initial vectors i_1, i_2, does $[\![i_1]\!] = [\![i_2]\!]$ hold?
- Language inclusion
  Given an order $\sqsubseteq$ and two initial vectors i_1, i_2, does $[\![i_1]\!] \sqsubseteq [\![i_2]\!]$ hold?
- Threshold/Universality
  Given an initial vector i and $T \in \mathbb{S}$, does $[\![i]\!] \sqsupseteq T$ hold?

Weighted Automaton over a Semiring



For the tropical semiring the order is $\sqsubseteq = \geq$

The automaton satisfies the threshold 3, i.e., every word has at most weight 3 (path $A \to B \to \cdots \to B \to C$).

Problems for Weighted Automata

What is known about these problems?

	equivalence	inclusion	threshold
arithmetic semiring	P [Tzeng]	undecidable	undecidable ($\geq$) [Paz]
tropical semiring	undecidable [Krob]	undecidable	PSPACE-cmpl. [Almagor,Boker,Kupferman]
distr. lattices	PSPACE-cmpl.	PSPACE-cmpl. [Kupferman,Lustig]	PSPACE-cmpl.

Up-To Techniques for NFAs

These problems are even PSPACE-complete for NFAs (lattice $\{0,1\}$, order $\sqsubseteq = \leq$).

Although these are fundamental problems for finite automata, there have only recently been major advances concerning efficiency:

- Antichain Algorithm [De Wulf,Doyen,Henzinger,Raskin, '06]
- Simulation Meets Antichains [Abdulla,Chen,Holík,Vojnar, '10]
- Up-To Techniques [Bonchi,Pous, '13]

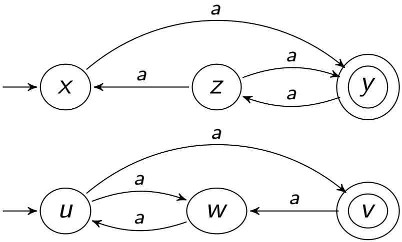
Up-To Techniques for NFAs

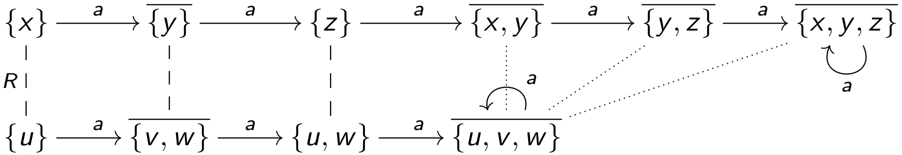
Checking Language Equivalence for NFAs

Find a bisimulation relation R on sets of states such that

- $S_1 \, R \, S_2$: the initial state sets are related
- Whenever $X_1 \, R \, X_2$, then $\delta_a(X_1) \, R \, \delta_a(X_2)$ for $a \in \Sigma$ (transfer property)
  ($\delta_a(X)$: successors of X under a)
- Whenever $X_1 \, R \, X_2$, then $X_1 \cap F_1 \neq \emptyset \iff X_2 \cap F_2 \neq \emptyset$
  (one set is accepting iff the other is accepting)

Up-To Techniques for NFAs





Up-To Techniques for NFAs

We can already stop at the pair $\{x, y\}, \{u, v, w\}$, since $\{x\}\, R\, \{u\}$, $\{y\}\, R\, \{v, w\}$ and $\{x, y\} = \{x\} \cup \{y\}$, $\{u, v, w\} = \{u\} \cup \{v, w\}$.

In the algorithm above we can write the transfer property as

- Whenever $X_1\, R\, X_2$, then $\delta_a(X_1)\, f(R)\, \delta_a(X_2)$

where $f(R)$ is

- the closure of R under union *or*
- the congruence closure $c(R)$ *or*
- $c(R \cup B)$ where B is a (pre-computed) bisimulation relation.

This is a so-called up-to technique, which has been studied extensively in process algebra [Milner,Sangiorgi,Pous]

Up-To Techniques for NFAs

Congruence closure $c(R)$: closure of R under equivalence and union

Given sets X, Y, how to decide whether $(X, Y) \in c(R)$?

- For each pair $(Z, Z') \in R$ define two rewriting rules $Z \mapsto Z \cup Z'$, $Z' \mapsto Z \cup Z'$.
- A rewriting rule $L \mapsto R$ can be applied to X whenever $L \subseteq X$ and then $X \rightsquigarrow X \cup R$ (X rewrites to $X \cup R$).
- $X \, c(R) \, Y$ iff X, Y rewrite to the same normal form.

Example:

$\{x\} \, R \, \{u\}$ generates rules $\{x\} \mapsto \{x, u\}$, $\{u\} \mapsto \{x, u\}$
$\{y\} \, R \, \{v, w\}$ generates rules $\{y\} \mapsto \{y, v, w\}$, $\{v, w\} \mapsto \{y, v, w\}$

$\{x, y\} \rightsquigarrow \{x, y, u\} \rightsquigarrow \{x, y, u, v, w\}$
$\{u, v, w\} \rightsquigarrow \{x, u, v, w\} \rightsquigarrow \{x, y, u, v, w\}$

Up-To Techniques for Weighted Automata

We adapt up-to techniques to weighted automata over ℓ-monoids.

ℓ-monoid

An ℓ-monoid $\mathbb{L}$ is a semiring, where the sum ($\oplus$) is a join operation ($\sqcup$).

Examples: tropical semiring, distributive lattices

Congruence Closure $c(R)$ for a relation R on vectors over $\mathbb{L}$

$$\frac{v\ R\ w}{v\ c(R)\ w} \qquad \frac{}{v\ c(R)\ v} \qquad \frac{v\ c(R)\ w}{w\ c(R)\ v}$$

$$\frac{u\ c(R)\ v \quad v\ c(R)\ w}{u\ c(R)\ w} \qquad \frac{v\ c(R)\ w}{s \otimes v\ c(R)\ s \otimes w} \quad \text{where } s \in \mathbb{L}$$

$$\frac{v_1\ c(R)\ v_1' \quad v_2\ c(R)\ v_2'}{v_1 \sqcup v_2\ c(R)\ v_1' \sqcup v_2'}$$

Up-To Techniques for Weighted Automata

We use a rewriting algorithm to decide $c(R)$, which is in general infinite:

How to decide whether $(v_1, v_2) \in c(R)$?

- For each pair $(v, v') \in R$, define two rewriting rules $v \mapsto v \sqcup v'$, $v' \mapsto v \sqcup v'$.
- A rewriting rule $\ell \mapsto r$ can be applied to w whenever $s \otimes \ell \sqsubseteq w$ for some $s \in \mathbb{L}$ and then $w \rightsquigarrow w \sqcup s \otimes r$.

  Better: $w \rightsquigarrow w \sqcup (\ell \rightarrow w) \otimes r$ where

  $$\ell \rightarrow w = \bigsqcup \{x \in \mathbb{L} \mid x \otimes \ell \sqsubseteq w\} \text{ (residuation)}$$
- $v_1\, c(R)\, v_2$ iff v_1, v_2 rewrite to the same normal form.

Up-To Techniques for Weighted Automata

Example for the tropical semiring (join $\sqcup$ is min, order $\sqsubseteq = \geq$)

- Relation:

$$R = \{\begin{pmatrix}\infty\\0\end{pmatrix}, \begin{pmatrix}0\\\infty\end{pmatrix}\}$$

- Rules:

$$\ell_1 = \begin{pmatrix}\infty\\0\end{pmatrix} \mapsto r_1 = \begin{pmatrix}0\\0\end{pmatrix}, \qquad \ell_2 = \begin{pmatrix}0\\\infty\end{pmatrix} \mapsto r_2 = \begin{pmatrix}0\\0\end{pmatrix}$$

- Rule application to $v = \binom{\infty}{3}$: $\ell_1 \to v = 3$ and

$$v = \begin{pmatrix}\infty\\3\end{pmatrix} \rightsquigarrow v \sqcup (\ell_1 \to v) \otimes r_1 = \begin{pmatrix}\infty\\3\end{pmatrix} \min \left(3 + \begin{pmatrix}0\\0\end{pmatrix}\right) = \begin{pmatrix}3\\3\end{pmatrix}$$

Up-To Techniques for Weighted Automata

$v_1 \, c(R) \, v_2$ iff v_1, v_2 rewrite to the same normal form (Theorem)

Prove that

- $v \rightsquigarrow w \;\Rightarrow\; v \, c(R) \, w$.
- Whenever $v \, c(R) \, w$, v can be rewritten to a vector larger (or equal) than w.
- Rewriting is confluent.
- Rewriting terminates: this holds for
  - the tropical semiring
    (natural numbers: Dickson's lemma; reals: more complex proof)
  - distributive lattices, under certain conditions

Language Equivalence for Weighted Automata

HKC (i_1, i_2) – Language Equivalence Check

```
(1) R is empty; todo is empty;
(2) insert (i1, i2) into todo;
(3) while todo is not empty do
 (3.1)  extract (v1', v2') from todo;
 (3.2)  if (v1', v2') ∈ c(R) then continue;
 (3.3)  if v1' · o ≠ v2' · o then return false ;
 (3.4)  for all a ∈ Σ,
          insert (v1' · Ta, v2' · Ta) into todo;
 (3.5)  insert (v1', v2') into R;
(4) return true;
```

HKC: Hopcroft-Karp with Congruence Closure

Language Inclusion for Weighted Automata

The algorithm can be adapted for language inclusion checks:

- Check $v_1' \cdot o \not\sqsubseteq v_2' \cdot o$ instead of $v_1' \cdot o \neq v_2' \cdot o$
- Compute $p(R)$ (precongruence closure instead of congruence closure)

  Remove symmetry rule and replace reflexivity rule by

  $$\frac{v \sqsubseteq v'}{v \; p(R) \; v'}$$

  Use a similar rewriting algorithm to decide $p(R)$.
- Additional optimization: replace $p(R)$ by $p(R \cup S)$ where S is a pre-computed simulation relation

Language Inclusion for Weighted Automata

HKP' (i_1, i_2) – Language Inclusion Check

```
(1) R is empty; todo is empty;
(2) insert (i1, i2) into todo;
(3) while todo is not empty do
 (3.1)  extract (v1', v2') from todo;
 (3.2)  if (v1', v2') ∈ p(R ∪ S) then continue;
 (3.3)  if v1'·o ⋢ v2'·o then return false ;
 (3.4)  for all a ∈ Σ,
           insert (v1'·Ta, v2'·Ta) into todo;
 (3.5)  insert (v1', v2') into R;
(4) return true;
```

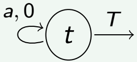
Threshold Problem for Weighted Automata

For the threshold problem we concentrate on the tropical semiring

Threshold Check

In order to show that the weights of all words are at most T for a given automaton:

- Perform a language inclusion check with the following automaton, using the up-to technique:



- In order to speed up termination replace all weights $> T$ by ∞ (abstraction $\mathcal{A}$, this is sound!)

Threshold Problem for Weighted Automata

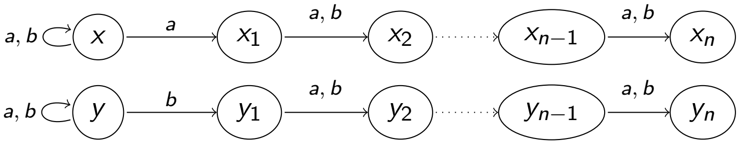
ABK(i) – Naive Algorithm (Threshold)

(1) $todo := \{i\}$;
(2) $P := \emptyset$;
(3) while $todo$ is not empty do
  (3.1) extract v from $todo$;
  (3.2) if $v \in P$ then continue ;
  (3.3) if $v \cdot o \not\leq T$ then return $false$;
  (3.4) for all $a \in \Sigma$ insert $\mathcal{A}(v \cdot T_a)$ into $todo$;
  (3.5) insert v into P ;
(4) return $true$;

ABK: Almagor, Boker, Kupferman

Threshold Problem for Weighted Automata

Example, where we have an exponential gain in the number of steps with the up-to technique:



Output weight is always 0, transition weight is always 1
Initial weight for x, y is 0, for all other states ∞

No threshold T is respected (a word of length m has weight m)

Threshold Problem for Weighted Automata

For ABK (naive algorithm), the runtime is exponential:

- every word w up to length n produces a different weight vector.
- For w with $|w| = m$ state x_i has weight m iff the i-last letter of the word is a, similarly state y_i has weight m iff the i-last letter is b.

Weights for aab:

x	x_1	x_2	x_3	x_4	$\dots$	y	y_1	y_2	y_3	y_4	$\dots$
3	∞	3	3	∞	$\dots$	3	3	∞	∞	∞	$\dots$

Threshold Problem for Weighted Automata

With HKP′ (up-to technique):

- we can deduce that x_i is simulated by x and y_i is simulated by y.
- With the rewriting rules every ∞-entry in x_i, y_i is replaced by m.

The above vector rewrites to:

x	x_1	x_2	x_3	x_4	$\dots$	y	y_1	y_2	y_3	y_4	$\dots$
3	3	3	3	3	$\dots$	3	3	3	3	3	$\dots$

All vectors for words of length m are in the precongruence relation: we keep only one representative.

Only linearly many words are considered!

Runtime Results on Randomly Generated Automata

We compared the following algorithms

- $\mathtt{HKP}'_{\mathcal{A}}$: language inclusion check (up-to) with abstraction and simulation relation
- $\mathtt{HKP}'_{\mathcal{A}}$: language inclusion check (up-to) with abstraction, without simulation relation
- $\mathtt{ABK}$: naive threshold algorithm

on randomly generated automata

- Alphabet size between 1 and 5
- Probability of an edge with weight unequal ∞: 90%
- If weight unequal ∞: random weight from $\{0, \dots, 10\}$

Runtime Results on Randomly Generated Automata

Threshold was respected in 14% of the cases.

We measured runtimes and list the 50%, 90% and 99% percentiles:

- 50% percentile: median
- 90% percentile: 90% of the runs were faster and 10% slower than the given time
- 99% percentile: analogously

We tested 1000 automata for each class $(|X|, T)$

Runtime Results on Randomly Generated Automata

		Runtime (millisec.)			Size of relation		
$(\lvert X\rvert, T)$	algo	50%	90%	99%	50%	90%	99%
(3,20)	$\mathtt{HKP}'_{\mathcal{A}}$	6	65	393	18	70	174
	$\mathtt{HKP}_{\mathcal{A}}$	4	64	466	18	71	192
	ABK	5	79	315	55	364	825
(6,20)	$\mathtt{HKP}'_{\mathcal{A}}$	239	7541	59922	111	589	1681
	$\mathtt{HKP}_{\mathcal{A}}$	234	7613	60360	111	589	1681
	ABK	253	16240	103804	702	6140	14126
(9,20)	$\mathtt{HKP}'_{\mathcal{A}}$	3885	168826	874259	407	2347	5086
	$\mathtt{HKP}_{\mathcal{A}}$	3838	168947	872647	407	2347	5086
	ABK	1744	301253	1617813	2171	22713	48735
(12,15)	$\mathtt{HKP}'_{\mathcal{A}}$	5127	363530	1971541	423	3001	6743
	$\mathtt{HKP}_{\mathcal{A}}$	5010	362908	1968865	423	3001	6743
	ABK	1418	509455	2349335	1672	27225	55627
(12,20)	$\mathtt{HKP}'_{\mathcal{A}}$	15101	789324	3622374	744	4489	9027
	$\mathtt{HKP}_{\mathcal{A}}$	15013	787119	3623393	744	4489	9027
	ABK	4169	1385929	4773543	3297	43756	80712

Runtime Results on Randomly Generated Automata

Observations:

- The up-to techniques have an advantage for the higher percentiles (90%, 99%), the naive technique is better for the lower percentiles (50%).
- The up-to techniques always shrink the relation substantially, the reductions in run-time are less substantial (overhead!).
- The use of simulation does not help for the randomly generated automata (since simulation relations are quite small).
  On the other hand they hardly slow down the runtime.

Conclusion

Related Work

- Some existing algorithms for language equivalence for weighted automata work up-to linear combinations [Sakarovitch], [Kiefer et al.], but not up-to congruence
- For fields (rings): $(v_1, v_2) \in c(R)$ iff $v_1 - v_2$ is in the subspace (submodule) generated by $\{w_1 - w_2 \mid (w_1, w_2) \in R\}$
- Few papers on language inclusion [Urabe,Hasuo]
- Up-to techniques for weighted automata have already been studied in a coalgebraic setting (abstract categorical framework) [Bonchi et al.], but without algorithms for deciding up-to congruence and without efficiency considerations

Conclusion

Future Work

- Find more efficient algorithms for the congruence check (rewriting algorithm) and the computation of the simulation relation
- More runtime results (with automata arising from case studies), benchmarks?
- Further case studies: distributive lattices