Decision Problems for Additive Regular Functions

Rajeev Alur  Mukund Raghothaman

University of Pennsylvania

Friday 12th July, 2013
What are we studying?

Regular functions
What are we studying?

**Regular functions**

Languages, $\Sigma^* \rightarrow \text{bool}$
What are we studying?

**Regular functions**

Languages, $\Sigma^* \rightarrow \text{bool}$

DFA
What are we studying?

**Regular functions**

<table>
<thead>
<tr>
<th>Languages, $\Sigma^* \rightarrow \text{bool}</th>
<th>DFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>String transductions, $\Sigma^* \rightarrow \Gamma^*</td>
<td>SST</td>
</tr>
</tbody>
</table>
What are we studying?

<table>
<thead>
<tr>
<th>Regular functions from $\Sigma^*$ to integers $\mathbb{Z}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages, $\Sigma^* \rightarrow \text{bool}$</td>
</tr>
<tr>
<td>String transductions, $\Sigma^* \rightarrow \Gamma^*$</td>
</tr>
<tr>
<td>Numerical functions, $\Sigma^* \rightarrow \mathbb{Z}$</td>
</tr>
</tbody>
</table>
Finite automata with cost labels, a la Mealy machines
Regular Functions
Modelling a coffee shop: Attempt 1

Finite automata with cost labels, a la Mealy machines

![Finite automata diagram]

- Start state
- Transitions labeled with cost labels (C/2, C/1, S, #)
- Intuitive, analyzable
- But not very expressive...
Regular Functions
Modelling a coffee shop: Attempt 1

Finite automata with cost labels, a la Mealy machines

- Intuitive, analyzable
Finite automata with cost labels, a la Mealy machines

- Intuitive, analyzable
- But not very expressive...
What if the survey gives us a discount for coffee already purchased?

- Not possible if costs are paid up front
- Cost of an event cannot be influenced by later events
What if the survey gives us a discount for coffee already purchased?

- Not possible if costs are paid up front
- Cost of an event cannot be influenced by later events

Solution?
What if the survey gives us a discount for coffee already purchased?

- Not possible if costs are paid up front
- Cost of an event cannot be influenced by later events

Solution? Registers!
Regular Functions / Cost Register Automata

Modelling a coffee shop: Attempt 2

\[
\begin{align*}
\text{start} & \quad q_s & \quad q_s \\
\# / y & := x & \quad \# / y & := x \\
S / x & := y & \quad S / x & := y \\
C / x & := x + 1 & \quad C / x & := x + 2 \\
\end{align*}
\]

\[
\begin{align*}
x & := x + 2 \\
y & := y + 1
\end{align*}
\]
Regular Functions / Cost Register Automata
Properties, or why they’re interesting

▶ Closure under linear combination, input reversal, etc.
▶ Fast equivalence procedure, decidable containment
▶ Equivalent to regular string-to-expression-tree transducers
Regular Functions / Cost Register Automata
Properties, or why they’re interesting

- Closure under linear combination, input reversal, etc. $f^{rev}$ defined as $f^{rev}(\sigma) = f(\sigma^{rev})$ is regular when $f$ is
- Fast equivalence procedure, decidable containment
- Equivalent to regular string-to-expression-tree transducers
Regular Functions / Cost Register Automata
Properties, or why they’re interesting

- Closure under linear combination, input reversal, etc.
  \( f^{rev} \) defined as \( f^{rev}(\sigma) = f(\sigma^{rev}) \) is regular when \( f \) is
- Fast equivalence procedure, decidable containment
- Equivalent to regular string-to-expression-tree transducers

\[ abbbaaa \ldots bba \]
Regular Functions / Cost Register Automata
Properties, or why they’re interesting

- Closure under linear combination, input reversal, etc.
  \( f^{\text{rev}} \) defined as \( f^{\text{rev}}(\sigma) = f(\sigma^{\text{rev}}) \) is regular when \( f \) is
- Fast equivalence procedure, decidable containment
- Equivalent to regular string-to-expression-tree transducers

\[
\begin{align*}
\text{abbbaa...bba} & \quad \rightarrow \\
\end{align*}
\]

- Connections to weighted automata
What are we studying?

Regular functions from $\Sigma^*$ to integers $\mathbb{Z}$

| Languages, $\Sigma^* \to \text{bool}$ | DFA |
| String transductions, $\Sigma^* \to \Gamma^*$ | SST |
| Numerical functions, $\Sigma^* \to \mathbb{Z}$ | ? |
What are we studying?
Cost register automata

Regular functions from $\Sigma^*$ to integers $\mathbb{Z}$
- Languages, $\Sigma^* \rightarrow \text{bool}$  
  DFA
- String transductions, $\Sigma^* \rightarrow \Gamma^*$  
  SST
- Numerical functions, $\Sigma^* \rightarrow \mathbb{Z}$  
  CRA
Motivating Question: How do we Compute the Register Complexity?
Motivating Question
Register complexity

Does the coffee shop CRA really need 2 registers?

```
C / x := x + 2
y := y + 1

S / x := y
C / x := x + 1
```

```
start

q-s
s

# / y := x

qs
x
```

```n
# / y := x
```

Register Separation
Register Separation
Confessions of a coffee addict

- Pick a large number, say $c = 1,000,000$
Register Separation
Confessions of a coffee addict

- Pick a large number, say $c = 1,000,000$
- Observe what happens after processing $C^c = CC \ldots C$

![Diagram showing register separation]

- $C$:
  - $x := x + 2$
  - $y := y + 1$

- $S$:
  - $x := y$
  - $y := x$
Register Separation
Confessions of a coffee addict

▶ Pick a large number, say $c = 1,000,000$
▶ Observe what happens after processing $C^c = CC \ldots C$
▶ $|x - y| \geq c$

- $C/x := x + 1$
- $S/x := y$
- $C/y := x$
- $S/y := x$
- $q/s$
- $qs$
Register Separation

- In general, for each $c$, there is a path to $q_s$ so $|x - y| \geq c$
- No 1-register machine can make up these arbitrary differences in finite time
Register Separation
Generalizing to $k$ registers

- Pick a state $q$, and $k$ registers
- Say, for each $c$, there is a string to $q$ so every pair is at least $c$ apart

Then $k$ registers are really necessary
Register Separation

Establishing the Converse
Register Separation
Establishing the converse

Claim
If the registers are not $k$-separable, then $k - 1$ registers suffice
Register Separation

Establishing the converse

Claim
If the registers are not $k$-separable, then $k - 1$ registers suffice

Separation: $\exists q, \forall c, \exists \sigma, \forall u, v, |u - v| \geq c$
Register Separation
Establishing the converse

Claim
If the registers are not $k$-separable, then $k - 1$ registers suffice

Non-Separation: $\forall q, \exists c, \forall \sigma, \exists u, v, |u - v| < c$
Register Separation
Establishing the converse

**Claim**
If the registers are not $k$-separable, then $k - 1$ registers suffice

**Non-Separation:** $\exists c, \forall q, \forall \sigma, \exists u, v, |u - v| < c$
Register Separation
Establishing the converse

Claim
If the registers are not \( k \)-separable, then \( k - 1 \) registers suffice

Non-Separation: \( \exists c, \forall \sigma, \forall q, \exists u, v, |u - v| < c \)
Claim
If the registers are not $k$-separable, then $k - 1$ registers suffice

Non-Separation: $\exists c, \forall \sigma, \forall q, \exists u, v, |u - v| < c$
Register Separation
Establishing the converse

Claim
If the registers are not \( k \)-separable, then \( k - 1 \) registers suffice

Non-Separation:  \( \exists c, \forall \sigma, \forall q, \exists u, v, |u - v| < c \)

\[
\exists c, \forall \sigma, \bigwedge_q \bigvee_{u,v} |u - v| < c
\]
Register Separation
Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge q \bigvee u,v \quad |u - v| < c \]
Register Separation

Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge_q \bigvee_{u,v} |u - v| < c \]

\[ (q, \langle u, v \rangle, d_{uv}) \]

- Says \( u - v = d_{uv} \), where \(-c < d_{uv} < c\)
- Wherever we see "v", replace with "u − d_{uv}"
Register Separation

Establishing the converse

\begin{align*}
\exists c, \forall \sigma, \bigwedge_{q} \bigvee_{u,v} |u - v| < c
\end{align*}

\[
(q, \langle u, v \rangle, d_{uv}) \quad \longrightarrow \quad (q', \langle \_, \_ \rangle, \_)
\]

- Says $u - v = d_{uv}$, where $-c < d_{uv} < c$
- Wherever we see "\( v \)" replace with "\( u - d_{uv} \)"
Register Separation

Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge_q \bigvee_{u,v} |u - v| < c \]

\[ (q, \langle u, v \rangle, d_{uv}) \rightarrow (q', \langle _, _ \rangle, _) \]

\[ (-c < u' - v' < c) \lor \ldots \]

- Says \( u - v = d_{uv} \), where \(-c < d_{uv} < c\)
- Wherever we see “\( v \)”, replace with “\( u - d_{uv} \)”
Register Separation

Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge_{q, u,v} \bigvee |u - v| < c \]

\[ u' := u'' + 2 \]

\[ v' := v'' + 3 \]

\[ (q, \langle u, v \rangle, d_{uv}) \rightarrow (q', \langle \_, \_ \rangle, \_ ) \]

\[ (-c < u' - v' < c) \lor \ldots \]

- Says \( u - v = d_{uv} \), where \(-c < d_{uv} < c\)
- Wherever we see “\( v \)”, replace with “\( u - d_{uv} \)”
Register Separation

Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge_q \bigvee_{u,v} |u - v| < c \]

\[ u' := u'' + 2 \]
\[ v' := v'' + 3 \]

\[ (q, \langle u, v \rangle, d_{uv}) \quad \xrightarrow{\quad} \quad (q', \langle \_ , \_ \rangle, \_ ) \]

\[ (-c + 1 < u'' - v'' < c) \lor \ldots \leftrightarrow (-c < u' - v' < c) \lor \ldots \]

➢ Says \( u - v = d_{uv} \), where \( -c < d_{uv} < c \)
➢ Wherever we see “v”, replace with “u - d_{uv}”
Register Separation
Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge_q \bigvee_{u,v} |u - v| < c \]

\[
\left( q, \left\{ \langle u, v \rangle, d_{uv} \right\}, \left\langle u'', v'' \right\rangle, d_{u''v''} \right) \quad \frac{u' := u'' + 2}{v' := v'' + 3} \rightarrow \left( q', \langle _, _, \rangle, _ \right)
\]

\((-c + 1 < u'' - v'' < c) \lor \ldots \leftarrow \quad \left(-c < u' - v' < c \right) \lor \ldots \)

- Inductive backpropagation!
- Invariants maintained in DNF form
Register Separation
Establishing the converse

\[ \exists c, \forall \sigma, \bigwedge_q \bigvee_{u,v} |u - v| < c \]

\[
\left( q, \left\{ \langle u, v \rangle, d_{uv}, d_{u''v''} \right\} \right) \quad \begin{align*}
u' &:= u'' + 2 \\
v' &:= v'' + 3
\end{align*}
\rightarrow (q', \langle u', v' \rangle, d_{u''v''} - 1)
\]

\[ (-c + 1 < u'' - v'' < c) \lor \ldots \leftarrow ( -c < u' - v' < c) \lor \ldots \]

- Inductive backpropagation!
- Invariants maintained in DNF form
Register Separation

Establishing the converse

INV(q) := INV(q) ∧ WP(INV(q'), τ)

Repeat at each transition τ until fixpoint

Claim
A fixpoint will eventually be reached
Register Separation
Establishing the converse

\[ \text{INV} (q) := \text{INV} (q) \land \text{WP} (\text{INV} (q'), \tau) \]

Repeat at each transition \( \tau \) until fixpoint
Register Separation
Establishing the converse

\[ \text{INV}(q) := \text{INV}(q) \land \text{WP}(\text{INV}(q'), \tau) \]

Repeat at each transition \( \tau \) until fixpoint

Claim
A fixpoint will eventually be reached
Register Separation

Final result

**Theorem**

*The register complexity is at least \(k\) iff the registers are \(k\)-separable*
Register Separation

Final result

**Theorem**

*The register complexity is at least k iff the registers are k-separable*

**Theorem**

*Computing the register complexity is PSPACE-complete*
Conclusion
Conclusion
What we talked about

- Described CRAs as a model for regular functions
- Introduced register separation in CRAs
-Outlined connection between separation and register complexity
Conclusion
What we didn’t talk about, i.e. what else is in the paper

- Machine-independent characterization of the register complexity
- Analysis of adversarial games over CRAs – optimal reachability
  Undecidable when domain is $\mathbb{Z}$
  EXPTIME-complete when domain is $\mathbb{N}$
- Proofs!
Conclusion

What's left to do

- Understanding register separation in models with binary addition, SSTs, etc.
- Optimal reachability in probabilistic variants
- Variants for $\omega$-strings / trees / ...
Thank you! Questions?
Reserve Slides
Reserve Slides

Gotchas
Gotchas

Bounded registers

- Definition engineering; claims remain true in spirit
- Consider hypothetical “constant-0” register
Gotchas
Domain of computation / Algebraic structure “+”

- Paper assumes $\mathbb{Z}$; also holds for $\mathbb{N}$
- Free algorithm for $\mathbb{Q}$: The rationals admit a notion of “GCD”

Conjecture

- *Similar results hold for $\mathbb{R}$ as well*
- *Can be easily generalized to any commutative group*
Reserve Slides

Weighted automata
Use non-determinism!
Use non-determinism!
Regular Functions / Cost Register Automata

Connection to weighted automata

- CRAs are equivalent to unambiguous WA
- CRA (min, +c) equivalent to (full) WA
  \[ x := \min(x, y), \ y := z + 3 \]
- Weighted automata are inherently non-deterministic
Fin!