From KAT expressions to automata

Marcello Bonsangue[1,2] Jan Rutten[1,3] Alexandra Silva[1]

[1]Centrum voor Wiskunde en Informatica
[2]LIACS - Leiden University
[3]Vrije Universiteit Amsterdam

Arco Meeting, May 2009

Motivation

- Efficient algorithms exist to compile regular expressions into automata (Berry-Sethi, Thompson, ...)
- Alternatives to the elegant Brzozowski algorithm
- Kozen recently extended Brzozowski algorithm to KAT

Can we extend the efficient algorithms to KAT as well?

Motivation

- Efficient algorithms exist to compile regular expressions into automata (Berry-Sethi, Thompson, ...)
- Alternatives to the elegant Brzozowski algorithm
- Kozen recently extended Brzozowski algorithm to KAT

Can we extend the efficient algorithms to KAT as well?

Recall : Berry-Sethi for RE

Basic idea: Assume that all occurrences of letters are different.

$$(ab+b)^*ba \to (a_1 b_2 + b_3)^* b_4 a_5$$

Definition

Let E be a regular expression and $\overline{E}$ the corresponding marked expression.

$$\begin{array}{lcl} \mathit{first}(E) & = & \{i \mid a_i w \in L(\overline{E})\} \\ \mathit{follow}(E,i) & = & \{j \mid u a_i a_j v \in L(\overline{E})\} \\ \mathit{last}(E) & = & \{i \mid w a_i \in L(\overline{E})\} \end{array}$$

The position automaton for E is defined as :

$$\mathcal{A}_{\mathbf{pos}}(E) = (\mathit{pos}(E), \Sigma, t_{\mathbf{pos}}, 0, \mathit{last}_0(E))$$

where

$$\mathit{last}_0(E) = \begin{cases} \mathit{last}(E) \cup \{0\} & \delta(E) = 1 \\ \mathit{last}(E) & \text{oth.} \end{cases} \quad t_{\mathbf{pos}} = \{(i,a,j) \mid j \in \mathit{follow}(E,i), a = \overline{a_j}\} \cup \{(0,a,j) \mid j \in \mathit{first}(E), a = \overline{a_j}\}$$

Recall : Berry-Sethi for RE

Basic idea: Assume that all occurrences of letters are different.

$$(ab+b)^* ba \rightarrow (a_1 b_2 + b_3)^* b_4 a_5$$

Definition

Let E be a regular expression and $\overline{E}$ the corresponding marked expression.

$$\begin{array}{lcl} \mathit{first}(E) & = & \{i \mid a_i w \in L(\overline{E})\} \\ \mathit{follow}(E,i) & = & \{j \mid u a_i a_j v \in L(\overline{E})\} \\ \mathit{last}(E) & = & \{i \mid w a_i \in L(\overline{E})\} \end{array}$$

The position automaton for E is defined as :

$$\mathcal{A}_{\mathbf{pos}}(E) = (\mathit{pos}(E), \Sigma, t_{\mathbf{pos}}, 0, \mathit{last}_0(E))$$

where

$$\mathit{last}_0(E) = \begin{cases} \mathit{last}(E) \cup \{0\} & \delta(E) = 1 \\ \mathit{last}(E) & \text{oth.} \end{cases} \quad t_{\mathbf{pos}} = \{(i,a,j) \mid j \in \mathit{follow}(E,i), a = \overline{a_j}\} \cup \{(0,a,j) \mid j \in \mathit{first}(E), a = \overline{a_j}\}$$

Algorithm

Berry and Sethi provided a algorithmic description to compute the follow sets.

Proposition

For a regular expression E, F(E, {!}), as defined below, yields pairs of the form $\langle i, follow(E!, i)\rangle$.

$$
\begin{array}{lcl}
F(E_1 + E_2, S) & = & F(E_1, S) \cup F(E_2, S) \\
F(E_1 E_2, S) & = & F(E_1, first(E_2) \cup \delta(E_2) \cdot S) \cup F(E_2, S) \\
F(E_1^*, S) & = & F(E_1, first(E_1) \cup S) \\
F(a_i, S) & = & \{\langle i, S\rangle\} \quad F(1, S) = \emptyset \quad F(0, S) = \emptyset
\end{array}
$$

! is just a trick to avoid computing last:
$i \in last(E) \iff ! \in follow(E!, i)$.

Example in the board – $(a_1 b_2 + b_3)^* b_4 a_5$.

Algorithm

Berry and Sethi provided a algorithmic description to compute the follow sets.

Proposition

For a regular expression E, F(E, {!}), as defined below, yields pairs of the form $\langle i, \mathit{follow}(E!, i)\rangle$.

$$
\begin{array}{lcl}
F(E_1 + E_2, S) & = & F(E_1, S) \cup F(E_2, S) \\
F(E_1 E_2, S) & = & F(E_1, \mathit{first}(E_2) \cup \delta(E_2) \cdot S) \cup F(E_2, S) \\
F(E_1^*, S) & = & F(E_1, \mathit{first}(E_1) \cup S) \\
F(a_i, S) & = & \{\langle i, S\rangle\} \quad F(1, S) = \emptyset \quad F(0, S) = \emptyset
\end{array}
$$

! is just a trick to avoid computing last:
$i \in \mathit{last}(E) \iff ! \in \mathit{follow}(E!, i)$.

Example in the board – $(a_1 b_2 + b_3)^* b_4 a_5$.

Algorithm

Berry and Sethi provided a algorithmic description to compute the follow sets.

Proposition

For a regular expression E, F(E, {!}), as defined below, yields pairs of the form $\langle i, \mathit{follow}(E!, i)\rangle$.

$$
\begin{array}{lcl}
F(E_1 + E_2, S) & = & F(E_1, S) \cup F(E_2, S) \\
F(E_1 E_2, S) & = & F(E_1, \mathit{first}(E_2) \cup \delta(E_2) \cdot S) \cup F(E_2, S) \\
F(E_1^*, S) & = & F(E_1, \mathit{first}(E_1) \cup S) \\
F(a_i, S) & = & \{\langle i, S\rangle\} \quad F(1, S) = \emptyset \quad F(0, S) = \emptyset
\end{array}
$$

! is just a trick to avoid computing last:
$i \in \mathit{last}(E) \iff\; ! \in \mathit{follow}(E!, i)$.

Example in the board – $(a_1 b_2 + b_3)^* b_4 a_5$.

KAT

$$Exp \ni e, f ::= p \in \Sigma \mid b \in BExp \mid e.f \mid e + f \mid e^*$$
$$BExp \ni b ::= t \in T \mid b_1.b_2 \mid b_1 + b_2 \mid \overline{b} \mid 0 \mid 1$$

$\mathcal{B} = Bexp/\equiv, \equiv$ logical equivalence, is a boolean algebra.

KAT

$$Exp \ni e, f ::= p \in \Sigma \mid b \in BExp \mid e.f \mid e + f \mid e^*$$
$$BExp \ni b ::= t \in T \mid b_1.b_2 \mid b_1 + b_2 \mid \overline{b} \mid 0 \mid 1$$

$\mathcal{B} = Bexp/\equiv, \equiv$ logical equivalence, is a boolean algebra.

KAT are regular expressions over $\Sigma \cup T$

Observation 1: We can apply the Berry-Sethi algorithm directly and get:

KAT

$$Exp \ni e, f ::= p \in \Sigma \mid b \in BExp \mid e.f \mid e + f \mid e^*$$
$$BExp \ni b ::= t \in T \mid b_1.b_2 \mid b_1 + b_2 \mid \overline{b} \mid 0 \mid 1$$

$\mathcal{B} = Bexp / \equiv, \equiv$ logical equivalence, is a boolean algebra.

KAT are regular expressions over $\Sigma \cup T$

Observation 1: We can apply the Berry-Sethi algorithm directly and get:

- A non-deterministic automaton

$$\mathcal{A}_e = (S, \delta : S \to 2 \times (\mathcal{P}S)^{\Sigma \cup T}), S \subseteq Exp$$

- $L(\mathcal{A}_e) = L(e)$

KAT

$$Exp \ni e, f ::= p \in \Sigma \mid b \in BExp \mid e.f \mid e + f \mid e^*$$
$$BExp \ni b ::= t \in T \mid b_1.b_2 \mid b_1 + b_2 \mid \overline{b} \mid 0 \mid 1$$

$\mathcal{B} = Bexp/\equiv, \equiv$ logical equivalence, is a boolean algebra.

KAT are regular expressions over $\Sigma \cup T$

Observation 1: We can apply the Berry-Sethi algorithm directly and get:

- A non-deterministic automaton

$$\mathcal{A}_e = (S, \delta : S \to 2 \times (\mathcal{P}S)^{\Sigma \cup T}), S \subseteq Exp$$

- $L(\mathcal{A}_e) = L(e)$
- $GS(\mathcal{A}) = GS(e)$ (due to Kozen). Good!

Examples

- $(b + p)qr^*$
- $b_1 + pb_2(b_3 + q)^*rb_4$

Can we do “better”?

- Less states

$S \to 2 \times (\mathcal{P}S)^{\Sigma \cup T}$ Original automata on GS, too many states

$S \to \mathcal{B} \times S^{\Sigma \times At}$ Deterministic version, less intuitive

Can we do "better"?

- Less states

$S \to 2 \times (\mathcal{P}S)^{\Sigma \cup T}$	Original automata on GS, too many states
$S \to \mathcal{B} \times (\mathcal{P}S)^{\Sigma \times \mathcal{B}}$	Compromise
$S \to \mathcal{B} \times S^{\Sigma \times At}$	Deterministic version, less intuitive

Can we do "better"?

- Less states

$$S \to 2 \times (\mathcal{P}S)^{\Sigma \cup T} \quad \text{Original automata on GS, too many states}$$

$$S \to \mathcal{B} \times (\mathcal{P}S)^{\Sigma \times \mathcal{B}} \quad \text{Compromise}$$

$$S \to \mathcal{B} \times S^{\Sigma \times At} \quad \text{Deterministic version, less intuitive}$$

At: minimal elements of $\mathcal{B}$ (truth assignments or valuations, $At \cong 2^T$).

Can we do "better"?

- Less states

$$S \to 2 \times (\mathcal{P}S)^{\Sigma \cup T} \quad \text{Original automata on GS, too many states}$$

$$S \to \mathcal{B} \times (\mathcal{P}S)^{\Sigma \times \mathcal{B}} \quad \text{Compromise}$$

$$S \to \mathcal{B} \times S^{\Sigma \times At} \quad \text{Deterministic version, less intuitive}$$

  At: minimal elements of $\mathcal{B}$ (truth assignments or valuations, $At \cong 2^T$).

- Example in the board.

Adapted Berry-Sethi for KAT

Basic idea: Assume that all occurrences of letters in Σ are different.

$$(b+p)qr^* \rightarrow (b+p_1)q_2r_3^*$$

Definition

Let e be a KAT expression and $\overline{e}$ the corresponding marked expression.

$$\begin{array}{lcl} first(e) & = & \{\langle b,i\rangle \mid b_1b_2\dots b_npx \in L(\overline{e})\,,\ b=\bigvee(b_1\wedge b_2\wedge\dots b_n)\} \\ follow(e,i) & = & \{\langle b,j\rangle \mid xp_ib_1b_2\dots b_nq_jy \in L(\overline{e})\,,\ b=\bigvee(b_1\wedge b_2\wedge\dots b_n)\} \\ last(e) & = & \{\langle b,i\rangle \mid xp_ib_1b_2\dots b_n \in L(\overline{e})\,,\ b=\bigvee(b_1\wedge b_2\wedge\dots b_n)\} \end{array}$$

The automaton corresponding to e is defined as :

$$\mathcal{A}_e = (pos(e), \Sigma, \langle o, \delta\rangle, 0)$$

where

$$o(i) = \begin{cases} E(e) & i=0 \\ b & \langle b,i\rangle \in last(e) \\ \bot & \text{otherwise} \end{cases} \qquad \begin{array}{c} \delta = \{\langle i, \langle b,p\rangle, j\rangle \mid \langle b,j\rangle \in follow(e,i), p=\overline{p_i}\} \\ \cup\ \{(0,\langle b,a\rangle,j) \mid \langle b,j\rangle \in first(e), a=\overline{a_j}\} \end{array}$$

Adapted Berry-Sethi for KAT

Basic idea: Assume that all occurrences of letters in Σ are different.

$$(b+p)qr^* \rightarrow (b+p_1)q_2r_3^*$$

Definition

Let e be a KAT expression and $\overline{e}$ the corresponding marked expression.

$$\begin{array}{lcl} \mathit{first}(e) & = & \{\langle b,i\rangle \mid b_1b_2\dots b_n px \in L(\overline{e})\,,\, b=\bigvee(b_1\wedge b_2\wedge\dots b_n)\} \\ \mathit{follow}(e,i) & = & \{\langle b,j\rangle \mid xp_ib_1b_2\dots b_nq_jy \in L(\overline{e})\,,\, b=\bigvee(b_1\wedge b_2\wedge\dots b_n)\} \\ \mathit{last}(e) & = & \{\langle b,i\rangle \mid xp_ib_1b_2\dots b_n \in L(\overline{e})\,,\, b=\bigvee(b_1\wedge b_2\wedge\dots b_n)\} \end{array}$$

The automaton corresponding to e is defined as :

$$\mathcal{A}_e = (pos(e), \Sigma, \langle o, \delta\rangle, 0)$$

where

$$o(i) = \begin{cases} E(e) & i=0 \\ b & \langle b,i\rangle \in \mathit{last}(e) \\ \perp & \text{otherwise} \end{cases} \qquad \begin{array}{l} \delta = \{\langle i,\langle b,p\rangle,j\rangle \mid \langle b,j\rangle \in \mathit{follow}(e,i), p=\overline{p_i}\} \\ \quad\cup\ \{(0,\langle b,a\rangle,j) \mid \langle b,j\rangle \in \mathit{first}(e), a=\overline{a_j}\} \end{array}$$

Adapted Berry-Sethi for KAT

Proposition

$GS(\mathcal{A}_e) = GS(e)$

Proposition

For a KAT expression e, $F(e, \{\langle \top, ! \rangle\})$, as defined below, yields pairs of the form $\langle i, follow(E!, i) \rangle$.

$$
\begin{array}{lcl}
F(e_1 + e_2, S) & = & F(e_1, S) \cup F(e_2, S) \\
F(e_1 e_2, S) & = & F(e_1, first(e_2) \cup E(e_2) \triangleleft S) \cup F(E_2, S) \\
F(e_1^*, S) & = & F(e_1, first(E_1) \cup S) \\
F(a_i, S) & = & \{\langle i, S \rangle\} \quad F(b, S) = \emptyset
\end{array}
$$

$b \triangleleft S = \{\langle b \wedge b', p \rangle \mid \langle b', p \rangle \in S\}$ (with $\perp \triangleleft S = \emptyset$)

Adapted Berry-Sethi for KAT

Proposition

$GS(\mathcal{A}_e) = GS(e)$

Proposition

For a KAT expression e, $F(e, \{\langle \top, ! \rangle\})$, as defined below, yields pairs of the form $\langle i, follow(E!, i)\rangle$.

$$\begin{array}{rcl} F(e_1 + e_2, S) & = & F(e_1, S) \cup F(e_2, S) \\ F(e_1 e_2, S) & = & F(e_1, first(e_2) \cup E(e_2) \lhd S) \cup F(E_2, S) \\ F(e_1^*, S) & = & F(e_1, first(E_1) \cup S) \\ F(a_i, S) & = & \{\langle i, S \rangle\} \quad F(b, S) = \emptyset \end{array}$$

$b \lhd S = \{\langle b \wedge b', p\rangle \mid \langle b', p \rangle \in S\}$ (with $\perp \lhd S = \emptyset$)

Examples revisited

- $(b + p)qr^*$
- $b_1 + pb_2(b_3 + q)^* rb_4$

Conclusions

- Extended Berry-Sethi's algorithm to KAT.
- New type of automaton acceptor of guarded strings.
- Still need complexity analysis.

Thanks!

Conclusions

- Extended Berry-Sethi's algorithm to KAT.
- New type of automaton acceptor of guarded strings.
- Still need complexity analysis.

Thanks!