le produzioni sono riconducibili alla forma $A \to aw$, dove a è un terminale e w è una sequenza potenzialmente vuota di variabili. Se si pone il limite che w contenga al più una variabile otteniamo la forma che vincola le grammatiche *Type-3*.

La motivazione originale per cui Chomsky introduce la sua gerarchia è che diversi aspetti del linguaggio naturale richiedono classi di grammatiche diverse per espressività. Ad esempio, la morfologia della lingua inglese (e quindi il vocabolario dei termini che si usano per costruire frasi) è modellabile tramite una grammatica meno espressiva di quanto invece necessario per caratterizzare la sintassi della lingua inglese (e quindi costruire frasi).[15] Per dare un esempio del legame tra grammatiche libere e le tipiche regole sintattiche del linguaggio naturale, consideriamo il seguente frammento:

$$\begin{aligned} \text{Frase} &\to \text{Soggetto Verbo Oggetto} \\ \text{Soggetto} &\to \textit{Alex} \mid \textit{Mary} \\ \text{Verbo} &\to \textit{mangia} \mid \textit{beve} \\ \text{Oggetto} &\to \textit{banane} \mid \textit{birra} \end{aligned}$$

dove i termini in corsivo rappresentano i terminali della grammatica, mentre gli altri termini rappresentano le variabili, di cui Frase è quella iniziale (la notazione $A \to \alpha_1 \mid \alpha_2$ è una forma contratta per $A \to \alpha_1$ e $A \to \alpha_2$). Quindi, in quattro passi di derivazione, dalla variabile iniziale Frase è possibile dedurre la sequenza di terminali *Alex beve birra*. Da notare che la grammatica descrive la sintassi del linguaggio, astraendo completamente dalla semantica.

[15] Nel primo caso basta una grammatica *Type-3*, mentre nel secondo occorre una grammatica *Type-2*.

Quanto ipotizzato a partire dalle idee di Chomsky, come l'asserzione secondo cui il cervello umano sarebbe naturalmente "programmato" per utilizzare un insieme di base di regole di riscrittura (la cosiddetta *grammatica universale*), va oltre la teoria dei linguaggi formali. Quel che è certo invece è lo stretto legame tra la teoria di Chomsky e, ad esempio, lo sviluppo dei linguaggi di programmazione.

Solo due anni dopo l'opera del 1957 di Chomsky, Backus [1959] presenta la grammatica (libera) per un linguaggio di programmazione, con l'obiettivo di formalizzarne la struttura sintattica e favorire quindi un procedimento automatico di compilazione. Un compilatore è un programma traduttore che trasforma un programma sorgente scritto in un linguaggio ad alto livello di astrazione (come Pascal, C, e Java) in un programma oggetto scritto in un linguaggio a più basso livello (come il linguaggio macchina). L'idea di un procedimento automatico di traduzione da un linguaggio "vicino" a quello naturale e comprensibile dall'uomo ad un linguaggio più simile alla logica dei circuiti digitali è stata fondamentale per lo sviluppo accelerato delle tecnologie informatiche. Nelle fasi iniziali del processo di traduzione, il compilatore effettua sul programma sorgente analisi di tipo lessicale, per la verifica dei termini usati, e di tipo sintattico, per la verifica della struttura delle istruzioni. Analogamente alle tesi di Chomsky su morfologia e sintassi del linguaggio naturale, gli strumenti richiesti per tali analisi sono diversi per espressività e tecniche usate. Parafrasando l'esempio di grammatica libera per un linguaggio naturale visto in precedenza, un frammento per un linguaggio di programmazione che descrive la sintassi di istruzioni condizionali e cicli può essere come segue:

$$\begin{aligned} \text{Statement} &\rightarrow \textit{if}\ \text{Boolean\_condition Statement}\ \textit{else}\ \text{Statement} \mid \\ &\quad\ \textit{while}\ \text{Boolean\_condition Statement} \\ \text{Boolean\_condition} &\rightarrow \textit{true} \mid \textit{false} \mid \ldots \end{aligned}$$

Il lavoro di Backus [1959] mette in evidenza le implicazioni della formalizzazione dei linguaggi cosiddetti liberi (ovvero generati da grammatiche libere).[16] Diventa quindi importante stabilire quale classe di automi sia necessaria per applicare l'approccio riconoscitivo allo studio di tali linguaggi. Purtroppo, la classe dei linguaggi regolari risulta essere propriamente inclusa nella classe dei linguaggi liberi, che quindi non sono rappresentabili tramite ASF. In questa lacuna, il limite degli ASF deriva dalla memoria finita, in quanto il numero di stati è fissato a priori e non dipende dinamicamente dalla lunghezza della stringa da riconoscere. Ad esempio, si consideri la grammatica libera $G_1 = \{\{S\}, \{a, b\}, S, \{S \rightarrow aSb \mid ab\}\}$, che genera $L(G_1) = \{a^n b^n \mid n > 0\}$, ovvero sequenze del simbolo a concatenate a sequenze di pari lunghezza del simbolo b. Un ASF riconoscitore dovrebbe avere una memoria in grado di "ricordare" il numero di occorrenze di a che vengono lette. Tuttavia, la memoria di un ASF sono gli stati, e per ricordarsi di aver letto a^n sarebbero necessari n stati, da cui segue l'assurdo dato che n è un intero grande a piacere mentre lo spazio degli stati dell'automa è limitato e fissato a priori. Quindi sorge spontanea la questione di capire quale sia effettivamente l'espressività degli ASF di Kleene. La risposta la troviamo in Chomsky e Miller [1958], dove si dimostra che la classe dei linguaggi riconosciuti da ASF coincide con la classe di linguaggi generati da grammatiche *Type-3* di Chomsky. Quindi, per trattare i linguaggi liberi dobbiamo

[16] Idee fondazionali erano già presenti in Turing [1937] e Post [1936].

introdurre una nuova classe di automi.

Gli automi a pila sono automi a stati finiti arricchiti con una memoria ausiliaria potenzialmente infinita, chiamata *stack* o *pila*, che può essere sia letta che scritta. Il modello viene definito da Ginsburg, Greibach, e Harrison [1967] come strumento formale per il controllo sintattico durante la compilazione di un programma. La caratteristica peculiare dello stack è quella di essere accessibile solamente in modalità *last-in-first-out*. Se paragoniamo lo stack ad una pila di piatti da lavare, tale politica di accesso coincide con la regola che userebbe un lavapiatti che ha cura per il servizio, ovvero prendendo o aggiungendo sempre il piatto in cima alla pila. In virtù della presenza dello stack, ogni transizione diventa funzione dello stato corrente, del prossimo simbolo della stringa da leggere e del simbolo presente in cima alla pila, mentre per effetto della transizione non solo l'automa cambia stato, ma il simbolo in cima alla pila viene sostituito con una nuova sequenza di simboli. Il criterio di riconoscimento di una stringa può essere determinato dal raggiungimento di uno stato di accettazione, come nel caso degli ASF, oppure *per pila vuota*, ovvero al termine della lettura della stringa la pila deve essere vuota. I due criteri risultano essere equivalenti. Non sono invece ugualmente espressivi automi a pila deterministici e nondeterministici, essendo i secondi strettamente più espressivi dei primi. Ad esempio, il nondeterminismo risulta indispensabile per riconoscere tramite automa a pila il linguaggio $L(G_1)$ precedentemente descritto. Vediamo come agirebbe tale automa nel riconoscere una generica stringa $w \in L(G_1)$. Nello stato iniziale, ad ogni mossa l'automa è disposto a leggere un simbolo a, copiarlo nello stack e rimanere in tale stato. In maniera nondeterministica, è disposto anche a transitare in un nuovo stato, dove l'automa

è disposto a leggere un simbolo b, prelevare un simbolo a dallo stack e rimanere in tale stato. Se al termine della lettura della stringa lo stack è vuoto, significa che sono state lette tante a quante b e la stringa viene accettata. Ovviamente, per come abbiamo definito gli automi nondeterministici, sappiamo che, tra le tante possibili, esiste una strategia che permette all'automa di "indovinare" il momento in cui transitare dal primo al secondo stato e quindi riconoscere la stringa.

Formalmente, un automa a pila nondeterministico è una tupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

dove, rispetto agli ingredienti già noti per gli ASF, abbiamo l'alfabeto Γ dei simboli per lo stack, tra cui si distingue il simbolo iniziale Z_0, che rappresenta il contenuto dello stack all'inizio della computazione.

La funzione di transizione è $\delta : (Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma) \to \mathcal{P}(Q \times \Gamma^*)$. Ad esempio, se $(p, \gamma) \in \delta(q, a, z)$, dalla configurazione in cui q è lo stato corrente, a il prossimo simbolo da leggere dalla stringa e z il simbolo in cima alla pila, l'automa può evolvere transitando nello stato p, leggendo a e scrivendo sulla pila la sequenza γ in sostituzione di z.

La classe dei linguaggi riconosciuti da automi a pila nondeterministici coincide con la classe dei linguaggi generati da grammatiche *Type-2* di Chomsky. A questo proposito, proponiamo al lettore interessato la traduzione più semplice, quella da una grammatica libera $G = (V, T, S, P)$ all'automa che riconosce $L(G)$ per pila vuota, definito come $M = (\{q\}, T, V \cup T, \delta, q, S)$. Lo stato è unico, con ruolo del tutto marginale, l'alfabeto è l'insieme dei terminali T della grammatica, la pila può ospitare variabili e terminali

(inizialmente contiene la variabile S da cui ogni derivazione parte), mentre non serve fissare un insieme di stati di accettazione. La funzione di transizione δ simula i passi di derivazione che dimostrano quali stringhe appartengono al linguaggio, utilizzando la pila come supporto di memoria temporaneo dove parcheggiare i risultati intermedi della derivazione. In particolare, per ogni variabile $A \in V$:

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \in P\}$$

ovvero se in testa alla pila c'è una variabile A allora la possiamo sostituire con β, simulando quindi l'applicazione della produzione $A \to \beta$. In tal caso nessun simbolo viene letto dalla stringa da riconoscere. Per ogni terminale $a \in T$:

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

ovvero se in testa alla pila c'è un terminale a (presente in quanto generato durante il procedimento di derivazione), lo confrontiamo con il prossimo simbolo da leggere nella stringa: se sono uguali il procedimento continua consumandoli entrambi, altrimenti la stringa viene rifiutata. Se il procedimento termina avendo letto tutta la stringa e svuotato la pila, allora il riconoscimento avviene con successo. Quindi, tutta l'espressività necessaria all'automa risiede nello stack. Questo schema di traduzione rappresenta il kernel di molti compilatori per moderni linguaggi di programmazione, dove l'analisi sintattica delle istruzioni rispetto alla grammatica del linguaggio avviene per opera di un modulo, chiamato *parser*, che implementa l'automa a pila riconoscitore del linguaggio.

Torniamo ora ai linguaggi liberi e alle loro complicazioni rispetto alle espressioni regolari. I linguaggi liberi non sono chiusi per intersezione e complemento, mentre lo sono per i tre operatori delle espressioni regolari. I problemi decidibili sono appartenenza, test del vuoto, finitezza ed infine equivalenza (limitatamente a linguaggi liberi riconosciuti da automi a pila deterministici), quest'ultimo dimostrato da Sénizergues [1997]. I problemi di indecidibilità sono molto più numerosi, tra cui citiamo equivalenza (per linguaggi liberi generici), test di intersezione vuota ($L_1 \cap L_2 = \emptyset$) e ambiguità. Il problema dell'ambiguità è particolarmente rilevante. Intuitivamente, una grammatica si dice ambigua quando esistono diverse deduzioni della stessa stringa, mentre un linguaggio si dice intrinsecamente ambiguo se ogni grammatica che lo genera è ambigua. È chiaro quindi che l'ambiguità causa problemi laddove si vuole automatizzare il procedimento di riconoscimento, in quanto deduzioni distinte potrebbero dar luogo a chiavi di lettura completamente diverse di una stringa, con conseguenze negative sulla sua interpretazione semantica. Sebbene scoraggiante da un punto di vista teorico, l'indecidibilità della ambiguità si aggira grazie a tecniche note per la definizione di produzioni la cui forma scongiura il pericolo. Ad esempio, le grammatiche libere sottostanti i moderni linguaggi di programmazione non sono ambigue e i relativi parser sono deterministici. Per completezza, va anche sottolineato che non esistono linguaggi regolari intrinsecamente ambigui.

La dimostrazione dei risultati negativi sopra citati si ottiene per riduzione da un noto problema indecidibile della teoria dei linguaggi formali, detto della corrispondenza di Post (cfr. Post [1946]), la cui definizione spicca per semplicità. Sono date due liste di stringhe $U = \alpha_1, \alpha_2, \ldots, \alpha_n$ e $V = \beta_1, \beta_2, \ldots, \beta_n$ definite su uno stesso alfabeto di

almeno due simboli. Il problema consiste nel decidere se esiste una sequenza di indici $i_1, \ldots, i_k$ tale che $\alpha_{i_1} \cdots \alpha_{i_k} = \beta_{i_1} \cdots \beta_{i_k}$, nel qual caso si dice che $(U; V)$ è una istanza con soluzione. Ad esempio, l'istanza $(a, ba, aab; aa, b, ab)$ ha una sua possibile soluzione nella parola $baaaaab$, ottenuta, sia per U che per V, componendone le stringhe secondo la sequenza di indici $2, 1, 1, 3$.

Mostriamo ora come questo problema indecidibile si possa ridurre al test di intersezione vuota tra linguaggi liberi. Date $U = \alpha_1, \alpha_2, \ldots, \alpha_n$ e $V = \beta_1, \beta_2, \ldots, \beta_n$ definite sull'alfabeto $\{a, b\}$, costruiamo due grammatiche libere sull'alfabeto di terminali $\{a, b, 1, \ldots, n\}$, G_U con produzioni:

$$A \to \alpha_1 A 1 \mid \ldots \mid \alpha_n A n \mid \alpha_1 1 \mid \ldots \mid \alpha_n n$$

e G_V con produzioni:

$$B \to \beta_1 B 1 \mid \ldots \mid \beta_n B n \mid \beta_1 1 \mid \ldots \mid \beta_n n$$

Rispetto al nostro esempio, si noti che la derivazione:

$$A \to baA2 \to baaA12 \to baaaA112 \to baaaaab3112$$

descrive la sequenza di stringhe di U da considerare per ottenere la parola $baaaaab$. Così come $baaaaab3112 \in L(G_U)$, è altrettanto dimostrabile che $baaaaab3112 \in L(G_V)$. Più in generale, se $L(G_U) \cap L(G_V) \neq \emptyset$, allora abbiamo una soluzione per l'istanza $(U; V)$,

ottenendo di fatto la riduzione da cui segue il risultato di indecidibilità.[17]

Chiudiamo il cerchio aperto con i sistemi di riscrittura di Thue osservando che l'approccio usato in Post [1946], dietro suggerimento di Church, viene successivamente usato in Post [1947] per stabilire l'indecidibilità di uno dei problemi fondamentali posti in Thue [1914], ovvero se tramite il suo sistema di riscrittura è possibile trasformare una certa stringa in un'altra data stringa. Il lavoro ha inizio così:

> Alonzo Church suggested to the writer that a certain problem of Thue [1914] might be proved unsolvable by the methods of Post [1946]. We proceed to prove the problem recursively unsolvable, that is, unsolvable in the sense of Church [1936], but by a method meeting the special needs of the problem.[18]

5. Verso i confini del calcolabile

Se modifichiamo gli automi a pila consentendo l'accesso a qualunque posizione dello stack, otteniamo una estensione nota come automa limitato linearmente. Più precisamente, la struttura di memoria ausiliaria diventa un nastro finito. Inizialmente il nastro contiene la stringa da riconoscere, delimitata sia a sinistra che a destra da due caratteri speciali, diversi tra loro, usati per riconoscere i "confini" del nastro. L'automa è posizionato sul primo simbolo della stringa e può leggere o scrivere sul nastro muovendosi in qualunque direzione, contrariamente agli ASF e a quelli a pila, che devono scandire la stringa da riconoscere da sinistra verso destra.

Storicamente, gli automi limitati linearmente non vengono proposti per estendere gli automi a pila, bensì per vincolare i gradi di libertà della MdT. Myhill [1960] osserva infatti che la MdT rappresenta un modello di calcolo puramente teorico e decide quindi di sviluppar-

[17] Se per assurdo $L(G_U) \cap L(G_V) \neq \emptyset$ fosse decidibile, allora per la riduzione appena vista lo sarebbe anche il problema della corrispondenza di Post.

[18] Cfr. Post [1947], p. 1.

ne una variante per calcolatori reali. Per questo scopo, è sufficiente limitare la dimensione del nastro a quella dell'input. L'automa definito da Myhill è deterministico e non viene messo in relazione con la teoria dei linguaggi formali. Tre anni più tardi, Landweber affronta il problema e dimostra che gli automi di Myhill riconoscono linguaggi dipendenti dal contesto, ovvero generati da grammatiche *Type-1* di Chomsky (cfr. Landweber [1963]). Poco più tardi, Kuroda [1964] introduce gli automi limitati linearmente nondeterministici e, seguendo la dimostrazione di Landweber, realizza che essi coincidono per espressività con le grammatiche *Type-1* di Chomsky. Se gli automi nondeterministici di Kuroda siano equivalenti a quelli deterministici di Myhill è tuttora un problema aperto, così come non è noto se esistano linguaggi dipendenti dal contesto intrinsecamente ambigui.

Formalmente, un automa limitato linearmente nondeterministico è una tupla:

$$M = (Q, \Sigma, B, \delta, q_0, X_l, X_r, F)$$

dove, agli ingredienti già noti, si aggiungono l'alfabeto B dei simboli su nastro (si noti che $\Sigma \subseteq B$) e i simboli speciali X_l e X_r usati per delimitare i confini sinistro e destro del nastro, rispettivamente. La funzione di transizione $\delta : (Q \times B) \to \mathcal{P}(Q \times B \times \{dx, sx\})$ determina, fissati stato e simbolo correnti, come questi cambiano e in quale direzione ci si sposta lungo il nastro. Per rispettare le limitazioni di movimento imposte dalla finitezza del nastro, se $(p, a, d) \in \delta(q, X_L)$ allora $a = X_L$ e $d = dx$, ovvero il delimitatore sinistro non può essere modificato e l'automa deve muoversi verso destra. Analogamente, se $(p, a, d) \in \delta(q, X_R)$ allora $a = X_R$ e $d = sx$. Il criterio di riconoscimento è stabilito dal

raggiungimento di una configurazione in cui l'automa si trova in uno stato finale.

I linguaggi dipendenti dal contesto sono chiusi per unione, concatenazione, chiusura di Kleene e, a differenza dei linguaggi liberi, per intersezione e complemento, quest'ultimo uno dei risultati della matematica discreta più importanti degli anni '80. Curiosamente, sebbene rimasto un problema aperto a lungo, è stato risolto nel 1988 in maniera indipendente e contemporanea da Immerman [1988] e da Szelepcsényi [1988]. L'unico problema decidibile rilevante è quello dell'appartenenza, risultato che, come vedremo, rende i linguaggi dipendenti dal contesto la più ampia classe di linguaggi decidibili.

Infine, vogliamo tornare al quesito che ci eravamo posti su quale classe di linguaggi formali fosse riconosciuta dal modello della MdT. Per trattare la MdT come un automa riconoscitore di stringhe, è sufficiente considerare la definizione formale di automa limitato linearmente e rilassare la condizione sulla finitezza del nastro.

Gli automi di Turing deterministici e nondeterministici hanno la stessa espressività e riconoscono una classe chiusa per unione, concatenazione, chiusura di Kleene, intersezione e priva di linguaggi intrinsecamente ambigui. Tale classe coincide con l'insieme dei linguaggi ricorsivamente enumerabili, ovvero la più ampia classe di linguaggi descrivibile tramite grammatiche, quelle *Type-0* secondo la gerarchia di Chomsky. Questa equivalenza fissa definitivamente il legame tra grammatiche, automi e linguaggi formali. Ovviamente, i limiti noti per la calcolabilità si propagano anche al problema del riconoscimento di linguaggi. Ad esempio, i linguaggi ricorsivamente enumerabili non sono decidibili, bensì solo semidecidibili: esiste un algoritmo che risponde affermativamente se $w \in L$, ma non fornisce alcuna risposta (ovvero non termina) se $w \notin L$. Esiste un più generale risultato

di indecidibilità che risale a Rice [1953], secondo cui le proprietà non banali degli insiemi ricorsivamente enumerabili (inclusi quindi i linguaggi riconosciuti da MdT) sono tutte indecidibili. In questo contesto, una proprietà non è banale se alcuni linguaggi della classe (ma non tutti) la soddisfano.

Concludendo, MdT e grammatiche generali fissano i confini della rappresentazione di linguaggi formali, mentre ASF (e automi a pila) segnano le basi per un approccio computazionale ai problemi interdisciplinari accennati all'inizio di questo lavoro.

6. NOTE CONCLUSIVE

Le numerose applicazioni tuttora attuali e la presenza di interessanti problemi aperti rendono la teoria degli automi un campo di ricerca vitale e prolifico. Esempi d'uso variano dalla elaborazione del linguaggio naturale (cfr. Roche e Schabes [1997]) ai compilatori per linguaggi di programmazione (cfr. Aho, Sethi, e Ullman [1986]) e alla biologia molecolare (cfr., Waterman [1995]). Per approfondire l'argomento senza cadere nei dettagli di specifici campi applicativi sono diversi i testi classici degni di nota. Tra i tanti, nel campo delle scienze informatiche due riferimenti chiari ed esaustivi sono Harrison [1978] e Hopcroft et al. [2007]. Sono più incentrati sulla teoria dei linguaggi formali ma decisamente più completi in questo contesto i contributi di Salomaa [1973] e [1981]. Per un approccio più filosofico, una interessante lettura è Novaes Dutilh [2012].

BIBLIOGRAFIA

Aho, A. V., Sethi, R., Ullman, J. D. (1986). *Compilers, principles, techniques and tools*. Addison Wesley.

Backus, J. W. (1959). "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference." In *International conference on information processing* (pp. 125–132).

Berstel, J. (1995). *Axel Thue's papers on repetitions in words: a translation* (Publications du LaCIM, 20). Université du Québec à Montrèal.

Büchi, R. (1960). "On a decision method in restricted second order arithmetic." In *Congress on logic, method, and philosophy of science* (pp. 1–12). Stanford University Press.

Chomsky, N. (1957). *Syntactic structures*. Mouton.

Chomsky, N., Miller, G. (1958). "Finite state languages." *Information and Computation, 1*, 91–112.

Chomsky, N., Schützenberger, M. P. (1963). "The algebraic theory of context-free languages." In P. Braffort D. Hirschberg (Eds.), *Computer programming and formal systems* (pp. 118–161). North Holland.

Church, A. (1936). "An unsolvable problem of elementary number theory." *American journal of mathematics, 58*, 345–363.

Davis, M. (1985). *Computability and unsolvability*. Dover Publications.

Frege, G. (1893). *Grundgesetze der Arithmetik*. Verlag Hermann Pohle. (Tr. inglese di: M. Furth (1964), *The Basics Laws of Arithmetic: Exposition of the System*, Berkeley University Press; P. Ebert, M. Rossberg e C. Wright (2013), *Gottlob Frege: Basic Laws of Arithmetic*, Oxford University Press.)

Ginsburg, S., Greibach, S., Harrison, M. (1967). "Stack automata and compiling."

Journal of the ACM, 14(1).

Gödel, K. (1931). "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme." *Monatshefte für Mathematik und Physik, 38*. (Tr. in. a cura di S. Feferman (1986), *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*, Oxford University Press.)

Greibach, S. (1981). "Formal languages: Origins and directions." *Annals History of Computation, 3*.

Harrison, M. A. (1978). *Introduction to formal language theory*. Addison Wesley.

Hilbert, D. (1928). "Die Grundlagen der Mathematik." In *Abhandlungen aus dem Seminar der Hamburgischen Universität, 6*. (Tr. in. a cura di J. van Heijenoort (1967), *From Frege to Gödel. A Source Book in Mathematical Logic, 1897–1931*, Harvard University Press.)

Hopcroft, J. E., Motwani, R., Ullman, J. D. (2007). *Introduction to automata theory, languages, and computation*. Addison Wesley.

Immerman, N. (1988). "Nondeterministic space is closed under complementation." *SIAM Journal of Computing, 17*, 275–303.

Jiang, T., Li, M., Ravikumar, B., Regan, K. W. (2010). "Formal grammars and languages." In *Algorithms and theory of computation handbook.* Chapman & Hall, CRC Princeton University Press.

Kleene, S. (1951). *Representation of events in nerve nets and finite automata* (Research Memorandum No. RM-704). U.S. Air Force.

Kleene, S. (1956). "Representation of events in nerve nets and finite automata." In

C. Shannon J. McCarthy (Eds.), *Automata studies* (pp. 3–42). Princeton University Press.

Kuroda, S.-Y. (1964). "Classes of languages and linear-bounded automata." *Information and Control, 7(2)*, 207–223.

Landweber, P. (1963). "Three theorems on phrase structure grammars of type 1." *Information and Control, 6(2)*, 131–136.

McCulloch, W., Pitts, W. (1943). "A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics, 5*, 115–133.

McNaughton, R., Yamada, H. (1960). "Regular expressions and state graphs for automata." *IEEE Transactions on Electronic Computers, 9(1)*, 39–47.

Myhill, J. (1960). *Linearly bounded automata* (Technical Note). Wright-Patterson Air Force Base.

Novaes Dutilh, C. (2012). *Formal languages in logic: A philosophical and cognitive analysis*. Cambridge University Press.

Post, E. L. (1936). "Finite combinatory processes." *Journal of Symbolic Logic, 1*, 103–105.

Post, E. L. (1946). "A variant of a recursively unsolvable problem." *Bulletin of American Mathematics Society, 52*, 264–268.

Post, E. L. (1947). "Recursive unsolvability of a problem of Thue." *Journal of Symbolic Logic, 12(1)*, 1–11.

Pressburger, M. (1929). "Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt." In *I con-*

grés de mathématiciens des pays Slaves (pp. 92–101). Warszawa. (Tr. in. di: R. Stansifer (1984), *Pressburger's Article on Integer Arithmetic: Remarks and Translation*, Cornell University.)

Rabin, M., Scott, D. (1959). "Finite automata and their decision problems." *IBM Journal of Research and Development, 3(2)*, 114–125.

Rice, H. G. (1953). "Classes of recursively enumerable sets and their decision problems." *Transactions of the American Mathematical Society, 74(2)*, 358–366.

Roche, E., Schabes, Y. (1997). *Finite-state language processing*. MIT Press.

Salomaa, A. (1973). *Formal languages*. Academic Press.

Salomaa, A. (1981). *Jewels of formal language theory*. Computer Science Press.

Schützenberger, M. P. (1965). "On finite monoids having only trivial subgroups." *Information and Control, 8*, 190–194.

Sénizergues, G. (1997). "The equivalence problem for deterministic pushdown automata is decidable." In *Automata, languages and programming* (Vol. LNCS 1256, pp. 671–681). Springer.

Szelepcsényi, R. (1988). "The method of forced enumeration for nondeterministic automata." *Acta Informatica, 26*, 279–284.

Thue, A. (1906). "Über unendliche Zeichenreihen." In *Norske Vid. Skrifter I Mat.-Nat. Kl., 7.* Christiania.

Thue, A. (1914). "Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln." In *Norske Vid. Skrifter I Mat.-Nat. Kl., 10.* Christiania.

Turing, A. (1937). "On computable numbers with an application to the

Entscheidungsproblem." In *London mathematical society, 42* (pp. 230–265).

Vardi, M., Wolper, P. (1986). "Automata-theoretic techniques for modal logics of programs." *Journal of Computer and System Sciences, 32(2)*, 183–221.

Waterman, M. S. (1995). *Introduction to computational biology*. Chapman and Hall.