Chapter 3

Regular grammars

3.1 Introduction

Other view of the concept of language:

- not the formalization of the notion of effective procedure,
- but set of words satisfying a given set of rules
- Origin : formalization of natural language.

Example

- a `phrase` is of the form `subject verb`
- a `subject` is a `pronoun`
- a `pronoun` is **he** or **she**
- a verb is **sleeps** or **listens**

Possible phrases:

1. **he listens**
2. **he sleeps**
3. **she sleeps**
4. **she listens**

Grammars

- Grammar: *generative* description of a language

- Automaton: *analytical* description

- Example: programming languages are defined by a grammar (BNF), but recognized with an analytical description (the parser of a compiler),

- Language theory establishes links between analytical and generative language descriptions.

3.2 Grammars

A grammar is a 4-tuple $G = (V, \Sigma, R, S)$, where

- V is an alphabet,
- $\Sigma \subseteq V$ is the set *terminal symbols* ($V - \Sigma$ is the set of *nonterminal symbols*),
- $R \subseteq (V^+ \times V^*)$ is a finite set of *production rules* (also called simply rules or productions),
- $S \in V - \Sigma$ is the *start symbol*.

Notation:

- Elements of $V - \Sigma$: $A, B, \ldots$
- Elements of Σ : $a, b, \ldots$.
- Rules $(\alpha, \beta) \in R$: $\alpha \to \beta$ or $\alpha \underset{G}{\to} \beta$.
- The start symbol is usually written as S.
- Empty word: ε.

Example :

- $V = \{S, A, B, a, b\}$,
- $\Sigma = \{a, b\}$,
- $R = \{S \to A, S \to B, B \to bB, A \to aA, A \to \varepsilon, B \to \varepsilon\}$,
- S is the start symbol.

Words generated by a grammar: example

$aaaa$ is in the language generated by the grammar we have just described:

S		
A	rule	$S \to A$
aA		$A \to aA$
aaA		$A \to aA$
$aaaA$		$A \to aA$
$aaaaA$		$A \to aA$
$aaaa$		$A \to \varepsilon$

Generated words: definition

Let $G = (V, \Sigma, R, S)$ be a grammar and $u \in V^+$, $v \in V^*$ be words. The word v *can be derived in one step from* u by G (notation $u \underset{G}{\Rightarrow} v$) if and only if:

- $u = xu'y$ (u can be decomposed in three parts x, u' and y ; the parts x and y being allowed to be empty),

- $v = xv'y$ (v can be decomposed in three parts x, v' and y),

- $u' \underset{G}{\rightarrow} v'$ (the rule (u', v') is in R).

Let $G = (V, \Sigma, R, S)$ be a grammar and $u \in V^+$, $v \in V^*$ be words. The word v *can be derived in several steps from* u (notation $u \overset{*}{\underset{G}{\Rightarrow}} v$) if and only if $\exists k \geq 0$ and $v_0 \dots v_k \in V^+$ such that

- $u = v_0$,
- $v = v_k$,
- $v_i \underset{G}{\Rightarrow} v_{i+1}$ for $0 \leq i < k$.

- Words generated by a grammar G: words $v \in \Sigma^*$ (containing only terminal symbols) such that

$$S \overset{*}{\underset{G}{\Rightarrow}} v.$$

- The language generated by a grammar G (written $L(G)$) is the set

$$L(G) = \{v \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} v\}.$$

Example :
The language generated by the grammar shown in the example above is the set of all words containing either only a's or only b's.

Types of grammars

Type 0: no restrictions on the rules.

Type 1: *Context sensitive* grammars.

The rules

$$\alpha \rightarrow \beta$$

satisfy the condition

$$|\alpha| \leq |\beta|.$$

Exception: the rule

$$S \rightarrow \varepsilon$$

is allowed as long as the start symbol S does not appear in the right hand side of a rule.

Type 2: *context-free* grammars.

Productions of the form

$$A \to \beta$$

where $A \in V - \Sigma$ and there is no restriction on β.

Type 3: *regular* grammars.

Productions rules of the form

$$\begin{aligned} &A \to wB \\ &A \to w \end{aligned}$$

where $A, B \in V - \Sigma$ and $w \in \Sigma^*$.

3.3 Regular grammars

Theorem:
A language is regular if and only if it can be generated by a regular grammar.

A. If a language is regular, it can be generated by a regular grammar.

If L is regular, there exists

$$M = (Q, \Sigma, \Delta, s, F)$$

such that $L = L(M)$. From M, one can easily construct a regular grammar

$$G = (V_G, \Sigma_G, S_G, R_G)$$

generating L.

G is defined by:

- $\Sigma_G = \Sigma$,
- $V_G = Q \cup \Sigma$,
- $S_G = s$,
- $R_G = \left\{ \begin{array}{ll} A \rightarrow wB, & \text{for all} (A, w, B) \in \Delta \\ A \rightarrow \varepsilon & \text{for all} A \in F \end{array} \right\}$

B. If a language is generated by a regular grammar, it is regular.

Let

$$G = (V_G, \Sigma_G, S_G, R_G)$$

be the grammar generating L. A nondeterministic finite automaton accepting L can be defined as follows:

- $Q = V_G - \Sigma_G \cup \{f\}$ (the states of M are the nonterminal symbols of G to which a new state f is added),
- $\Sigma = \Sigma_G$,
- $s = S_G$,
- $F = \{f\}$,
- $\Delta = \left\{ \begin{array}{ll} (A, w, B), & \text{for all} A \rightarrow wB \in R_G \\ (A, w, f), & \text{for all} A \rightarrow w \in R_G \end{array} \right\}.$

3.4 The regular languages

We have seen four characterizations of the regular languages:

1. regular expressions,
2. deterministic finite automata,
3. nondeterministic finite automata,
4. regular grammars.

Properties of regular languages

Let L_1 and L_2 be two regular languages.

- $L_1 \cup L_2$ is regular.
- $L_1 \cdot L_2$ is regular.
- L_1^* is regular.
- L_1^R is regular.
- $\overline{L_1} = \Sigma^* - L_1$ is regular.
- $L_1 \cap L_2$ is regular.

$L_1 \cap L_2$ regular ?

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Alternatively, if $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ accepts L_1 and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ accepts L_2, the following automaton, accepts $L_1 \cap L_2$:

- $Q = Q_1 \times Q_2$,
- $\delta((q_1, q_2), \sigma) = (p_1, p_2)$ if and only if $\delta_1(q_1, \sigma) = p_1$ and $\delta_2(q_2, \sigma) = p_2$,
- $s = (s_1, s_2)$,
- $F = F_1 \times F_2$.

- Let Σ be the alphabet on which L_1 is defined, and let $\pi : \Sigma \to \Sigma'$ be a function from Σ to another alphabet Σ'.

  This fonction, called a *projection function* can be extended to words by applying it to every symbol in the word, *i.e.* for $w = w_1 \ldots w_k \in \Sigma^*$, $\pi(w) = \pi(w_1) \ldots \pi(w_k)$.

  If L_1 is regular, the language $\pi(L_1)$ is also regular.

Algorithms

Les following problems can be solved by algorithms for regular languages:

- $w \in L$?
- $L = \emptyset$?
- $L = \Sigma^*$? $\qquad (\overline{L} = \emptyset)$
- $L_1 \subseteq L_2$? $\qquad (\overline{L_2} \cap L_1 = \emptyset)$
- $L_1 = L_2$? $\qquad (L_1 \subseteq L_2$ and $L_2 \subseteq L_1)$

3.5 Beyond regular languages

- Many languages are regular,

- But, all languages cannot be regular for cardinality reasons.

- We will now prove, using another techniques that some specific languages are not regular.

Basic Observations

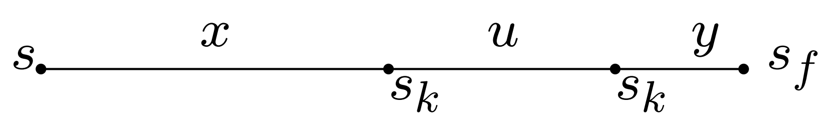
1. All finite languages (including only a finite number of words) are regular.

2. A non regular language must thus include an infinite number of words.

3. If a language includes an infinite number of words, there is no bound on the size of the words in the language.

4. Any regular language is accepted by a finite automaton that has a given number number m of states.

5. Consider an infinite regular language and an automaton with m states accepting this language. For any word whose length is greater than m, the execution of the automaton on this word must go through an identical state s_k at least twice, a nonempty part of the word being read between these two visits to s_k.



6. Consequently, all words of the form xu^*y are also accepted by the automaton and thus are in the language.

The "pumping" lemmas (theorems)

First version

Let L be an infinite regular language. Then there exists words $x, u, y \in \Sigma^*$, with $u \neq \varepsilon$ such that $xu^n y \in L\ \forall n \geq 0$.

Second version :

Let L be a regular language and let $w \in L$ be such that $|w| \geq |Q|$ where Q is the set of states of a determnistic automaton accepting L. Then $\exists x, u, y$, with $u \neq \varepsilon$ and $|xu| \leq |Q|$ such that $xuy = w$ and, $\forall n$, $xu^n y \in L$.

Applications of the pumping lemmas

The langage

$$a^n b^n$$

is not regular. Indeed, it is not possible to find words x, u, y such that $xu^k y \in a^n b^n \ \forall k$ and thus the pumping lemma cannot be true for this language.

$u \in a^*$: impossible.

$u \in b^*$: impossible.

$u \in (a \cup b)^* - (a^* \cup b^*)$: impossible.

The language

$$L = a^{n^2}$$

is not regular. Indeed, the pumping lemma (second version) is contradicted.

Let $m = |Q|$ be the number of states of an automaton accepting L. Consider a^{m^2}. Since $m^2 \geq m$, there must exist x, u and y such that $|xu| \leq m$ and $xu^n y \in L\ \forall n$. Explicitly, we have

$$\begin{array}{ll} x = a^p & 0 \leq p \leq m-1, \\ u = a^q & 0 < q \leq m, \\ y = a^r & r \geq 0. \end{array}$$

Consequently $xu^2y \notin L$ since $p + 2q + r$ is not a perfect square. Indeed,

$$m^2 < p + 2q + r \leq m^2 + m < (m+1)^2 = m^2 + 2m + 1.$$

The language

$$L = \{a^n \mid n \text{ is prime}\}$$

is not regular. The first pumping lemma implies that there exists constants p, q and r such that $\forall k$

$$xu^k y = a^{p+kq+r} \in L,$$

in other words, such that $p + kq + r$ is prime for all k. This is impossible since for $k = p + 2q + r + 2$, we have

$$p + kq + r = \underbrace{(q + 1)}_{>1}\underbrace{(p + 2q + r)}_{>1},$$

Applications of regular languages

Problem : To find in a (long) character string w, all ocurrences of words in the language defined by a regular expression α.

1. Consider the regular expression $\beta = \Sigma^*\alpha$.
2. Build a nondeterministic automaton accepting the language defined by β
3. From this automaton, build a *deterministic* automaton A_β.
4. Simulate the execution of the automaton A_β on the word w. Whenever this automaton is in an accepting state, one is at the end of an occurrence in w of a word in the language defined by α.

Applications of regular languages II: handling arithmetic

- A number written in base r is a word over the alphabet $\{0, \ldots, r-1\}$ ($\{0, \ldots, 9\}$ in decimal, $\{0, 1\}$ en binary).

- The number represented by a word $w = w_0 \ldots w_l$ is $nb(w) = \sum_{i=0}^{l} r^{l-i} nb(w_i)$

- Adding leading 0's to the representation of a number does not modify the represented value. A number thus has a infinite number of possible representations. Number encodings are read most significant digit first, and all possible encodings will be taken into account.

- **Exemple:** The set of binary representations of 5 is the language 0^*101.

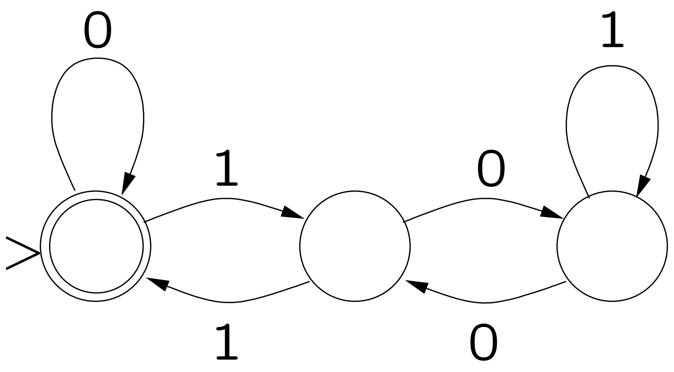
Which sets of numbers can be represented by regular languages?

- Finite sets.

- The set of multiples of 2 is represented by the language $(0 \cup 1)^*0$.

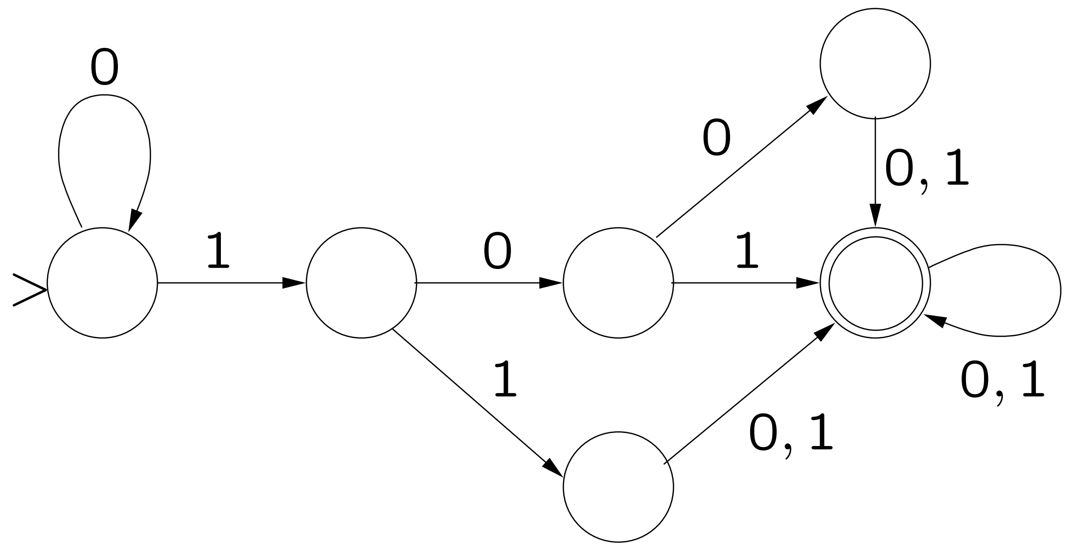
- The set of powers of 2 is represented by the language 0^*10^*, but is not representable in base 3.

- The set of multiples of 3 is represented by the following automaton.



Set of numbers represented by regular languages (continued)

- The set of numbers $x \geq 5$ is represented by the automaton



- More generally, one can represent sets of the form $\{ax \mid x \in N\}$ or $\{x \geq a \mid x \in N\}$ for any given value of a.

Set of numbers represented by regular languages (continued II)

- Combining the two types of sets: sets of the form $\{ax + b \mid x \in N\}$, for any given a and b.

- Union of such sets: the *ultimately periodic sets*.

- Intersection and complementation add nothing more.

- The only sets that can be represented in all bases are the ultimately periodic sets.

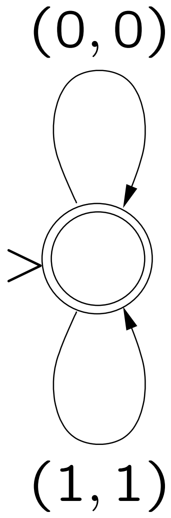
Representing vectors of numbers

- Each number is represented by a word, and bits in identical positions are read together.

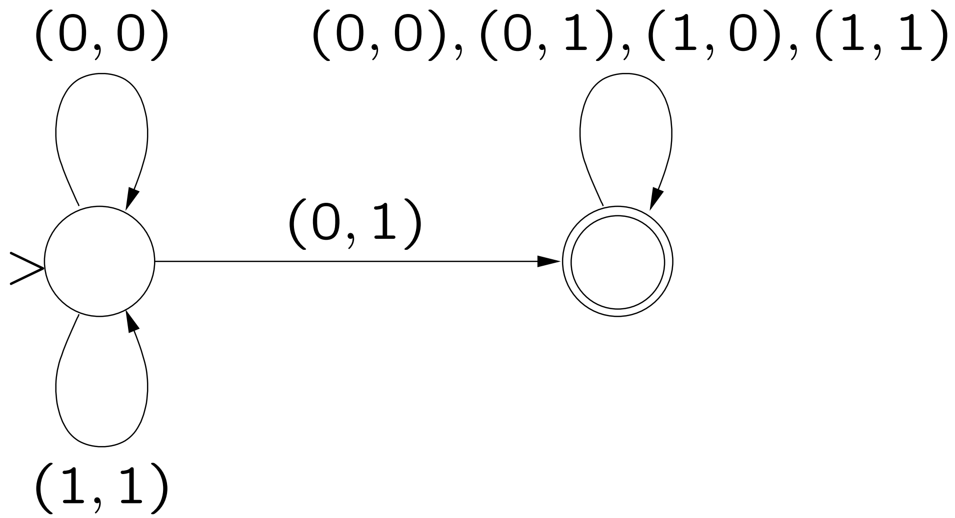
- **Example:**
  - the vector $(5,9)$ is encoded by the word $(0,1)(1,0)(0,0)(1,1)$ defined over the alphabet $\{0,1\} \times \{0,1\}$.
  - The set of binary encodings of the vector $(5,9)$ is $(0,0)^*(0,1)(1,0)$ $(0,0)(1,1)$.

Which sets of number vectors can be represented by regular languages?

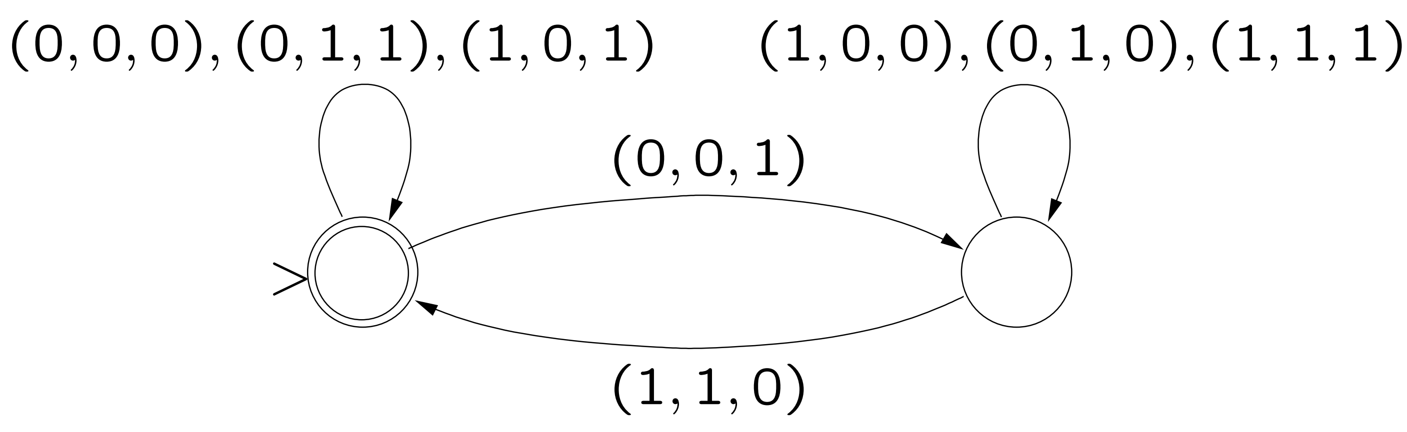
- The set of binary encodings of the vectors (x, y) such that $x = y$ is accepted by the automaton



- Vectors (x, y) such that $x < y$



- Three-dimentional vectors (x, y, z) such that $z = x + y$



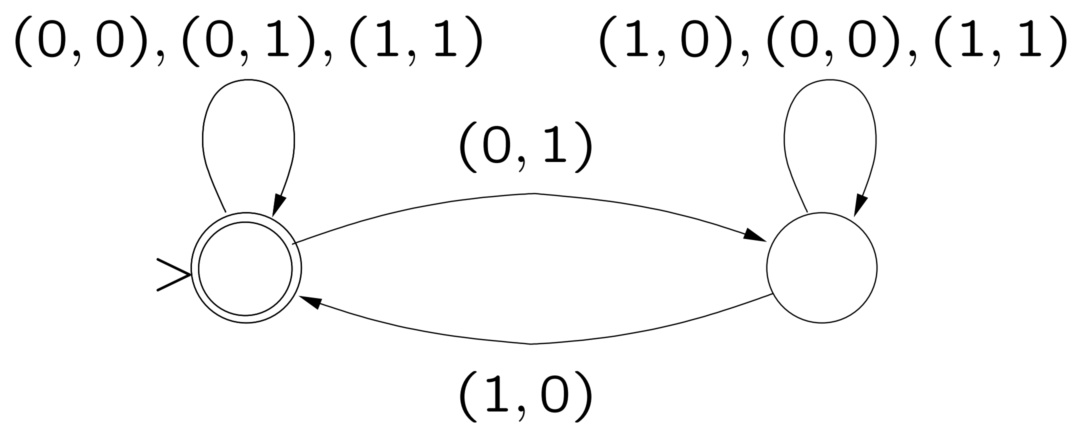
Definable sets of number vectors (continued)

- Intersection, union, complement of representable sets (closure properties of regular languages).

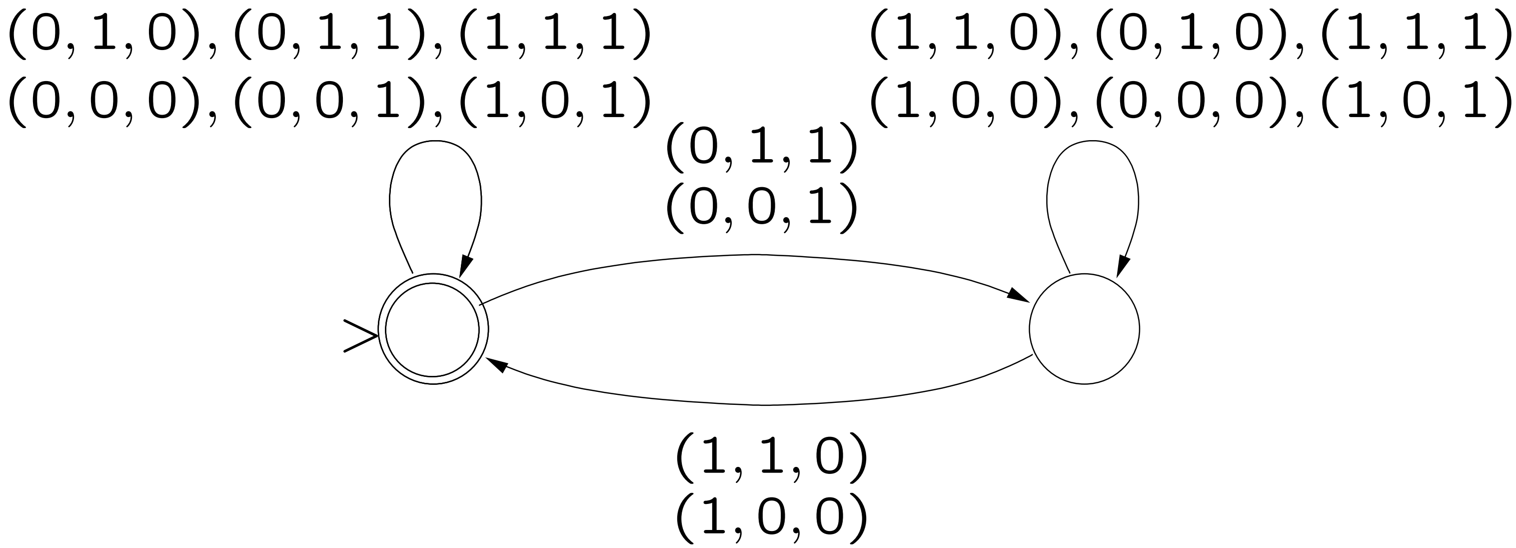
- Modifying the number of dimensions: projection and the inverse operation.

- **Remark:** projection does not always preserve the determinism of the automaton.

- **Example:** $\{(x,z) \mid \exists y \; x+y=z\}$ $(x \leq z)$.



- Adding a dimension to the previous automaton yields



- which is not equivalent to the automaton to which projection was applied.

Representable sets of vectors: conclusions

- Linear equality and inequality constraints

- **Example:** an automaton for $x + 2y = 5$ can be obtained by combing the automata for the following constraints:

$$\begin{aligned}
z_1 &= y \\
z_2 &= y + z_1 \\
z_3 &= x + z_2 \\
z_3 &= 5.
\end{aligned}$$

- There exists also a more direct construction.

Representable vector sets: conclusions (continued)

- Boolean combinations of linear constraints

- Existential quantification can be handled with projection ($\exists x$).

- For universal quantification, one uses $\forall x f \equiv \neg\exists\neg f$

- **Example:** It is possible to build an automaton accepting the representations of the vectors (x, y) satisfying the arithmetic constraint

$$\forall u \exists t[(2x + 3y + t - 4u = 5) \vee (x + 5y - 3t + 2u = 8)]$$

- This is Presburger arithmetic, which corresponds exactly to the sets representable by automata in all bases.