Automata-based decision procedures

IAM, Lecture 4

Lukáš Holík

Reminder: Presburger arithmetic

Is interpreted over $\mathbb{N}$, has the signature

$$\{0, S, +, =\}$$

Part I

Formulae as automata

Numbers as words

- in last significant bit first encoding (LSBF)

0	is encoded as	0
1	is encoded as	1
2	is encoded as	01
10	is encoded as	0101

...

Numbers as words

- in last significant bit first encoding (LSBF)

$$\begin{array}{rl} 0 & \text{is encoded as } 0 \\ 1 & \text{is encoded as } 1 \\ 2 & \text{is encoded as } 01 \\ 10 & \text{is encoded as } 0101 \\ & \dots \end{array}$$

- also, every word from $w0^*$ denotes the same number as w

010
0100
01000 all encode 2.
01000000000
...

Assignments as words

- assignments seen as k-tuples of numbers
  (+ an ordering on its k free variables)

$$\llbracket 2x = y \rrbracket = \{(0,0),(1,2),(2,4),\ldots\}$$

Assignments as words

- assignments seen as k-tuples of numbers
  (+ an ordering on its k free variables)

$$[\![2x = y]\!] = \{(0,0),(1,2),(2,4),\ldots\}$$

- encoded as words over the alphabet $\{0,1\}^k$

Assignments as words

- assignments seen as k-tuples of numbers (+ an ordering on its k free variables)

$$[\![2x = y]\!] = \{(0,0),(1,2),(2,4),\ldots\}$$

- encoded as words over the alphabet $\{0,1\}^k$
- for $k = 2$, the alphabet is $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 1 \end{smallmatrix} \right\}$

Assignments as words

- assignments seen as k-tuples of numbers
  (+ an ordering on its k free variables)

$$[\![2x = y]\!] = \{(0,0),(1,2),(2,4),\ldots\}$$

- encoded as words over the alphabet $\{0,1\}^k$
- for $k = 2$, the alphabet is $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 1 \end{smallmatrix} \right\}$
- and the assignment $\{x \mapsto 2, y \mapsto 4\}$, i.e. $(2,4)$, is encoded as

$$\begin{matrix} 010 \\ 001 \end{matrix}$$

Assignments as words

- assignments seen as k-tuples of numbers
  (+ an ordering on its k free variables)

$$[\![2x = y]\!] = \{(0,0),(1,2),(2,4),\ldots\}$$

- encoded as words over the alphabet $\{0,1\}^k$
- for $k = 2$, the alphabet is $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 1 \end{smallmatrix} \right\}$
- and the assignment $\{x \mapsto 2, y \mapsto 4\}$, i.e. $(2,4)$, is encoded as

$$\begin{matrix} 010 \\ 001 \end{matrix}$$

- and all the other words in $w(0^2)^*$

$$\begin{matrix} 010\,0 \\ 001\,0 \end{matrix}, \begin{matrix} 010\,00 \\ 001\,00 \end{matrix}, \begin{matrix} 010\,000 \\ 001\,000 \end{matrix}, \ldots$$

Assignments as words

- assignments seen as k-tuples of numbers
  (+ an ordering on its k free variables)

$$[\![2x = y]\!] = \{(0,0),(1,2),(2,4),\ldots\}$$

- encoded as words over the alphabet $\{0,1\}^k$
- for $k = 2$, the alphabet is $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 0 \\ 1 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 0 \end{smallmatrix}, \begin{smallmatrix} 1 \\ 1 \end{smallmatrix} \right\}$
- and the assignment $\{x \mapsto 2, y \mapsto 4\}$, i.e. $(2,4)$, is encoded as

$$\begin{matrix} 010 \\ 001 \end{matrix}$$

- and all the other words in $w(0^2)^*$

$$\begin{matrix} 010\,0 \\ 001\,0 \end{matrix}, \begin{matrix} 010\,00 \\ 001\,00 \end{matrix}, \begin{matrix} 010\,000 \\ 001\,000 \end{matrix}, \ldots$$

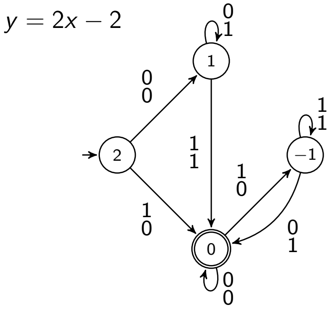
- $L(\varphi)$ denotes all encodings of all satisfying assignments of φ

Formulae as automata

- Presburger formulae can be translated to automata that accept exactly all encodings of their satisfying assignments.

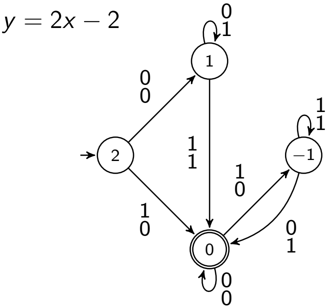
Formulae as automata

- Presburger formulae can be translated to automata that accept exactly all encodings of their satisfying assignments.



Formulae as automata

- Presburger formulae can be translated to automata that accept exactly all encodings of their satisfying assignments.



- To decide satisfiability of a formula φ
  - construct an automaton A with $L(A) = L(\varphi)$
  - and test emptiness of its language.

Deciding Presburger using automata

- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.

$$\neg\,(x \geq y) \;\wedge\; \exists z.\,(z \leq x+4 \vee \exists w.\, x < w < y)$$

Deciding Presburger using automata

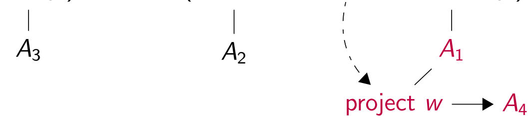
- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.

$$\neg \underbrace{(x \geq y)}_{A_3} \wedge \exists z.\, (\underbrace{z \leq x+4}_{A_2} \vee \exists w.\, \underbrace{x < w < y}_{A_1})$$

Deciding Presburger using automata

- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.

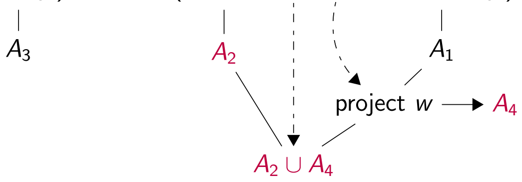
$$\neg\,(x \geq y) \;\wedge\; \exists z.\,(z \leq x + 4 \vee \exists w.\, x < w < y)$$



Deciding Presburger using automata

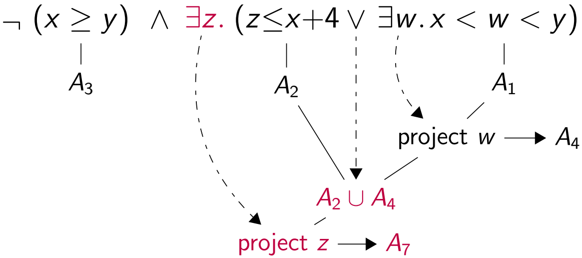
- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.

$$\neg\,(x \geq y) \;\wedge\; \exists z.\,(z \leq x+4 \vee \exists w.\, x < w < y)$$



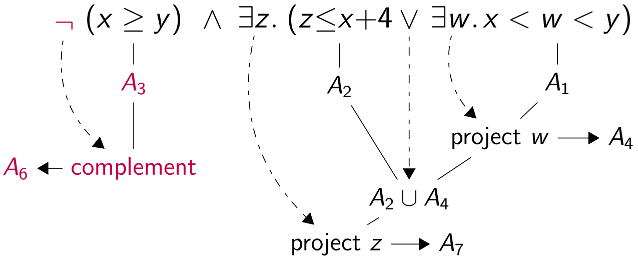
Deciding Presburger using automata

- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.



Deciding Presburger using automata

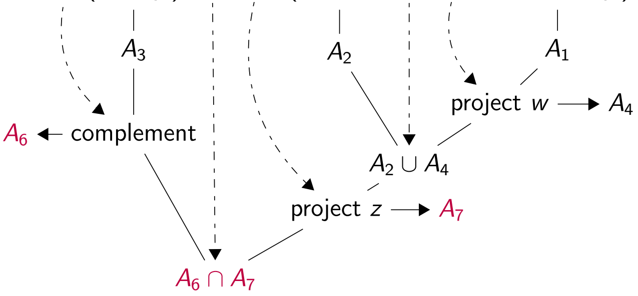
- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.



Deciding Presburger using automata

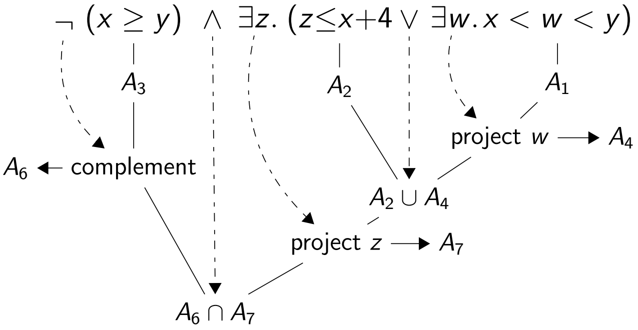
- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.

$$\neg\,(x \geq y) \;\wedge\; \exists z.\,(z \leq x+4 \vee \exists w.\, x < w < y)$$



Deciding Presburger using automata

- Build the FA with $L(A) = L(\varphi)$ inductively to φ's structure.



- Then check language emptiness of $A = A_6 \cap A_7$.

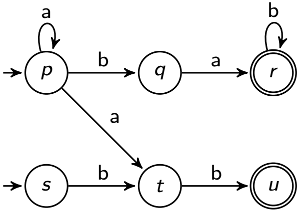
Ingredients

1. automata for atomic predicates
2. automata constructions for $\cup, \cap, \neg, \exists$
3. automata language emptiness test

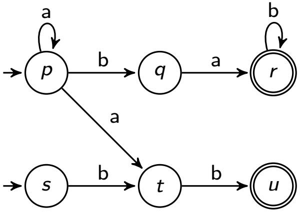
Part II

Automata crash course

Finite automaton $A = (\Sigma, Q, I, \delta, F)$

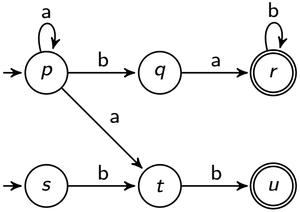


Finite automaton $A = (\Sigma, Q, I, \delta, F)$



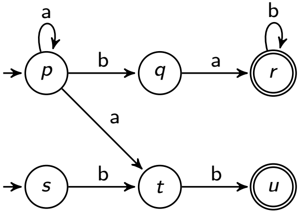
- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$

Finite automaton $A = (\Sigma, Q, I, \delta, F)$



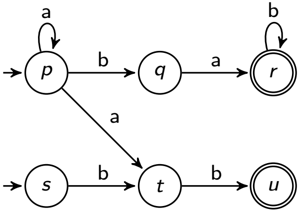
- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$
- transition function $\delta : Q \times \Sigma \to 2^Q$.

Finite automaton $A = (\Sigma, Q, I, \delta, F)$



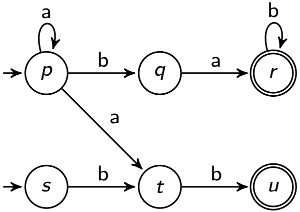
- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$
- transition function $\delta : Q \times \Sigma \to 2^Q$.
- a word is read in a run, which can be accepting or rejecting

Finite automaton $A = (\Sigma, Q, I, \delta, F)$



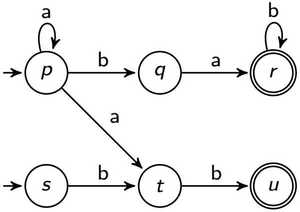
- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$
- transition function $\delta : Q \times \Sigma \to 2^Q$.
- a word is read in a run, which can be accepting or rejecting
- accepts a word if it has some accepting run over it

Finite automaton $A = (\Sigma, Q, I, \delta, F)$



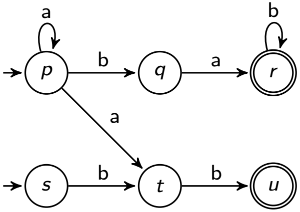
- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$
- transition function $\delta : Q \times \Sigma \to 2^Q$.
- a word is read in a run, which can be accepting or rejecting
- accepts a word if it has some accepting run over it
- language $L(A)$ is the set of all accepted words

Finite automaton $A = (\Sigma, Q, I, \delta, F)$



- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$
- transition function $\delta : Q \times \Sigma \to 2^Q$.
- a word is read in a run, which can be accepting or rejecting
- accepts a word if it has some accepting run over it
- language $L(A)$ is the set of all accepted words
- can be deterministic or nondeterministic

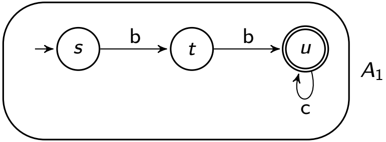
Finite automaton $A = (\Sigma, Q, I, \delta, F)$

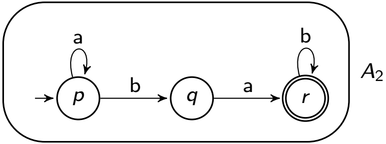


- finite sets: alphabet Σ, states Q, initial states $I \subseteq Q$, final/accepting states $F \subseteq Q$
- transition function $\delta : Q \times \Sigma \to 2^Q$.
- a word is read in a run, which can be accepting or rejecting
- accepts a word if it has some accepting run over it
- language $L(A)$ is the set of all accepted words
- can be deterministic or nondeterministic
  - deterministic has at most one run for every word

Automata union, $\cup$

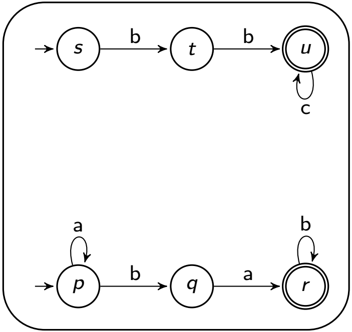
- We need $L(A_1) \cup L(A_2) = L(A_1 \cup A_2)$
- Simply unite the automata.





Automata union, $\cup$

- We need $L(A_1) \cup L(A_2) = L(A_1 \cup A_2)$
- Simply unite the automata.



$A_1 \cup A_2$

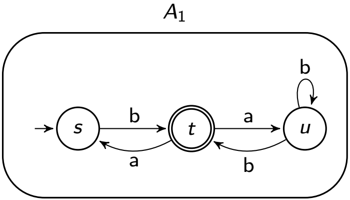
Assume $Q_1 \cap Q_2 = \emptyset$.
$Q' = Q_1 \cup Q_2$
$I' = I_1 \cup I_2$
$F' = F_1 \cup F_2$
$\delta' = \delta_1 \cup \delta_2$

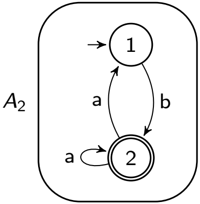
Automata intersection, $\cap$

- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

Automata intersection, $\cap$

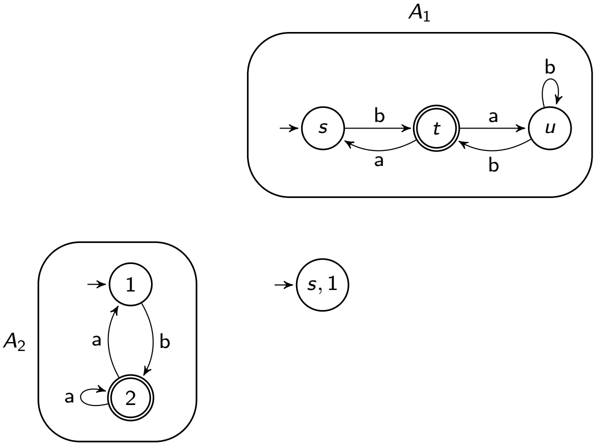
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.





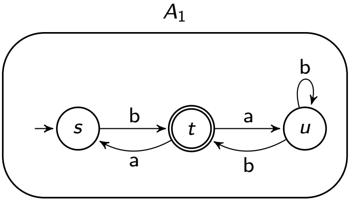
Automata intersection, $\cap$

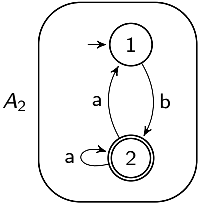
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

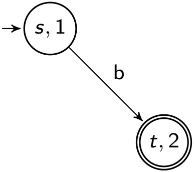


Automata intersection, $\cap$

- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

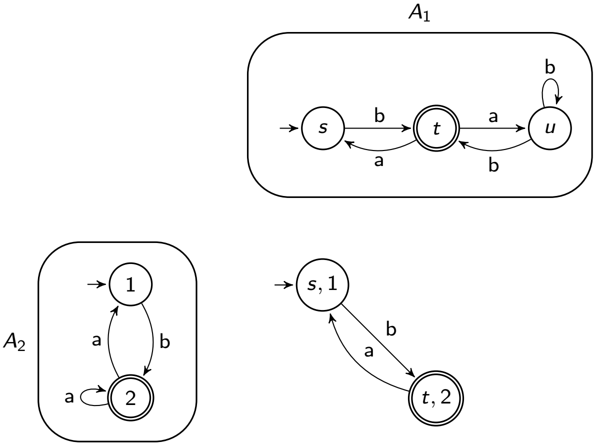






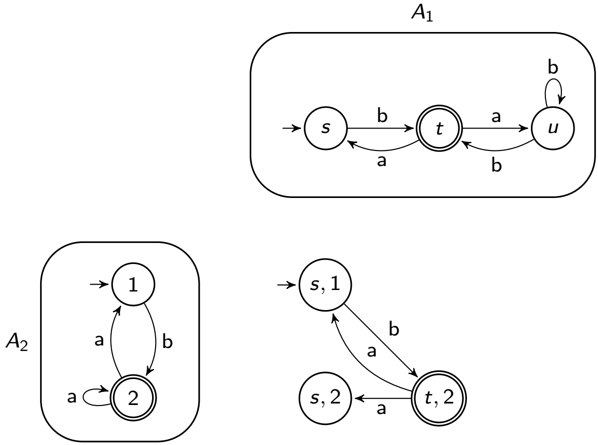
Automata intersection, $\cap$

- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.



Automata intersection, $\cap$

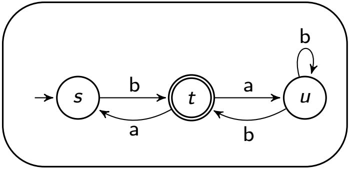
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.



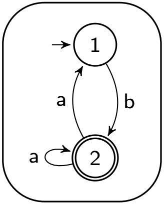
Automata intersection, $\cap$

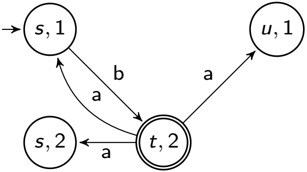
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

A_1



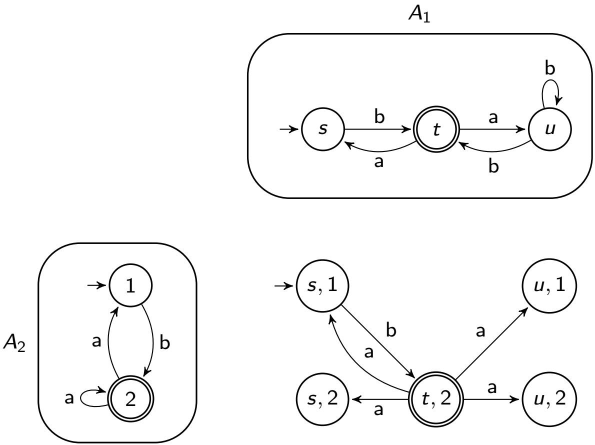
A_2





Automata intersection, $\cap$

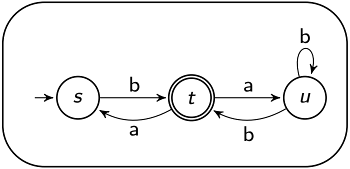
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.



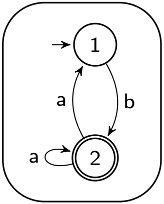
Automata intersection, $\cap$

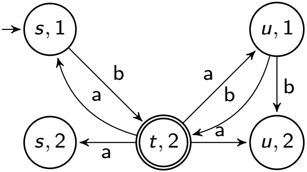
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

A_1



A_2



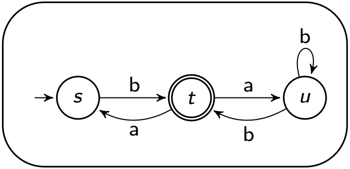


Automata intersection, $\cap$

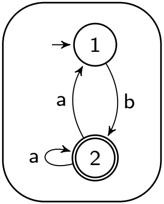
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

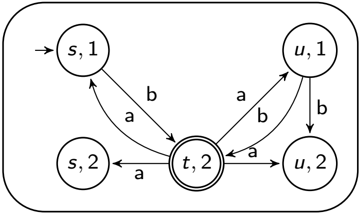
$$
\begin{aligned}
Q' &= Q_1 \times Q_2 \\
I' &= I_1 \times I_2 \\
F' &= F_1 \times F_2 \\
\delta'((q, r), a) &= \delta(q, a) \times \delta(r, a)
\end{aligned}
$$

A_1



A_2





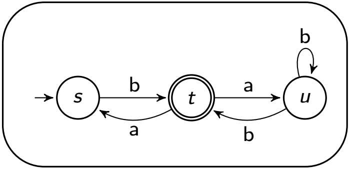
$A_1 \cap A_2$

Automata intersection, $\cap$

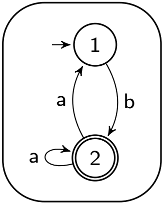
- We need $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.
- Use product construction.

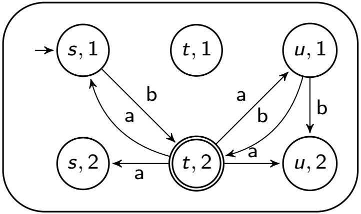
$$Q' = Q_1 \times Q_2$$
$$I' = I_1 \times I_2$$
$$F' = F_1 \times F_2$$
$$\delta'((q, r), a) = \delta(q, a) \times \delta(r, a)$$

A_1



A_2

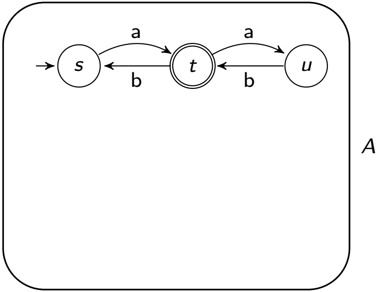




$A_1 \cap A_2$

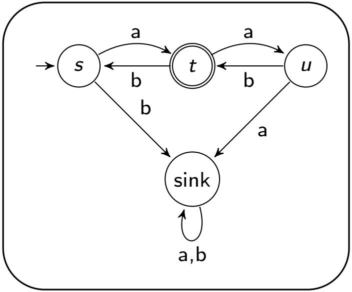
Automata complement, $\neg$

- We need $\Sigma^* \setminus L(A) = L(\neg A)$
- If deterministic, complete and negate acceptance.



Automata complement, $\neg$

- We need $\Sigma^* \setminus L(A) = L(\neg A)$
- If deterministic, complete and negate acceptance.

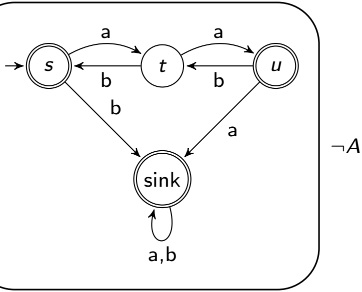


A completed

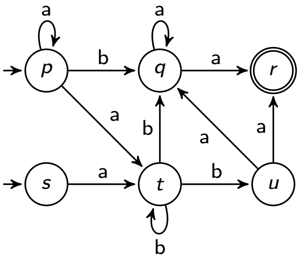
Automata complement, $\neg$

- We need $\Sigma^* \setminus L(A) = L(\neg A)$
- If deterministic, complete and negate acceptance.

Assume A determ. complete.
$Q' = Q$
$I' = I$
$F' = Q \setminus F$
$\delta' = \delta$



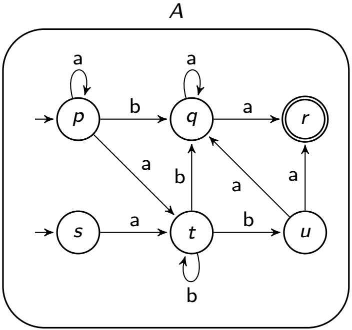
Complement has a problem with nondeterminism



- accepting as well as rejecting runs over *aba*
- hence *aba* is in $L(A)$ and stays after negating acceptance
- determinisation is needed

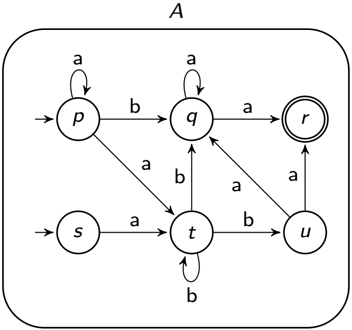
Determinization

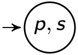
- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.



Determinization

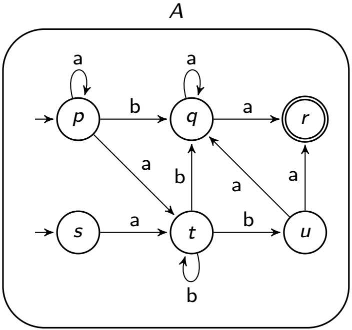
- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

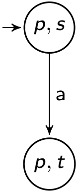




Determinization

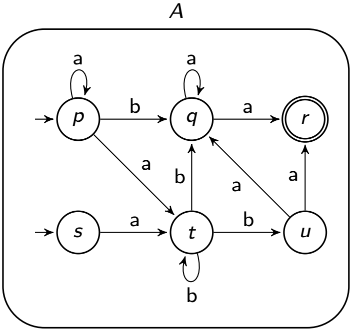
- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

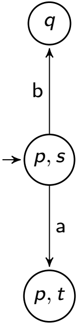




Determinization

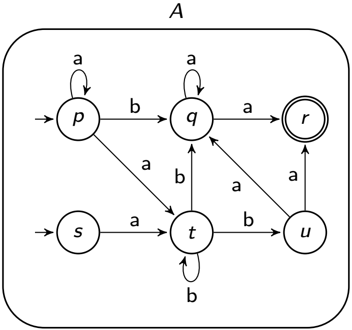
- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

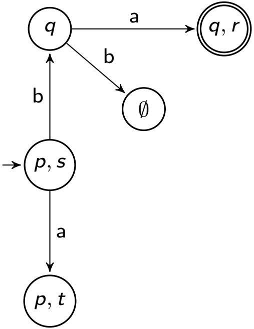




Determinization

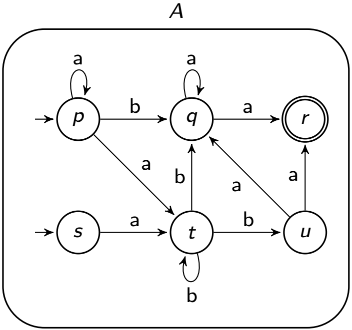
- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

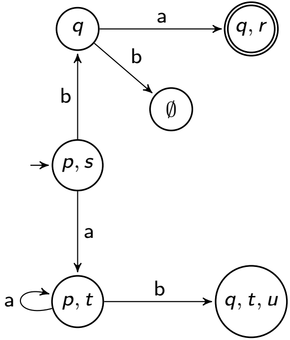




Determinization

- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

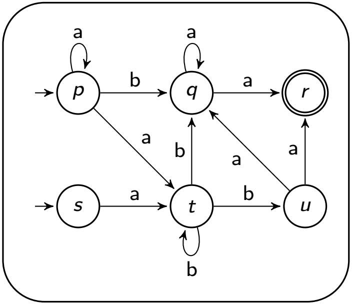


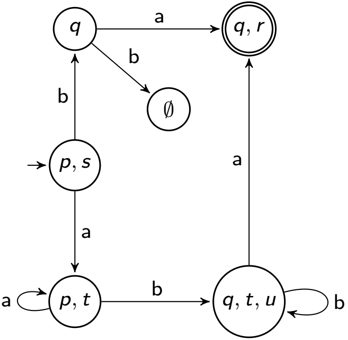


Determinization

- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

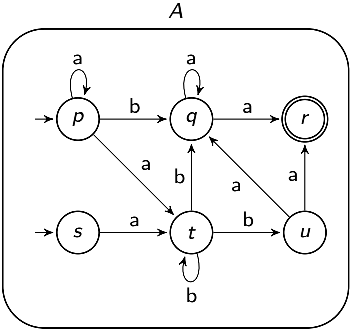
A

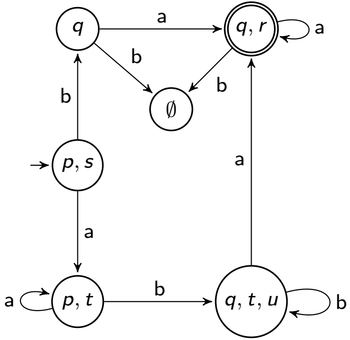




Determinization

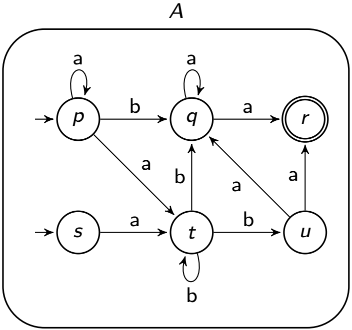
- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

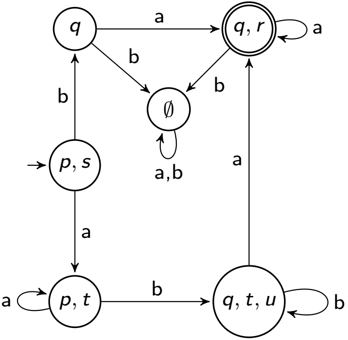




Determinization

- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

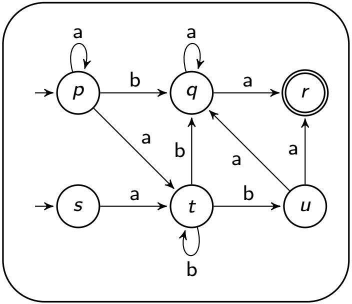




Determinization

- We need a deterministic A' with $L(A') = L(A)$.
- Subset construction.

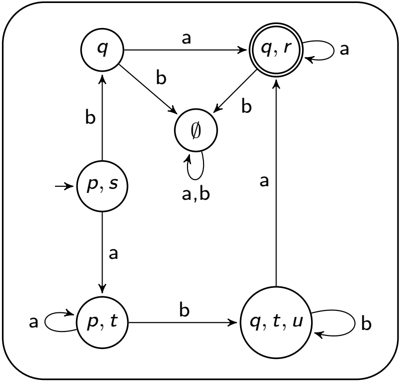
A



$Q' = 2^Q$ $\quad F' = \{S \in Q' \mid S \cap F \neq \emptyset\}$

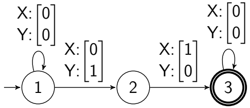
$I' = \{I\}$ $\quad \delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$

A' = determinized A



Automata projection, $\exists x$

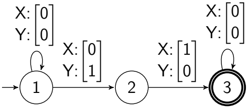
- Remove the x track (project on the y track).



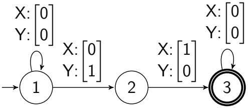
A

Automata projection, $\exists x$

- Remove the x track (project on the y track).



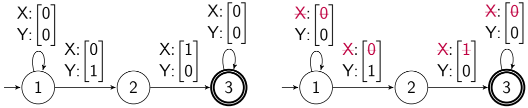
A



$\rightarrow$ Projection on X

Automata projection, $\exists x$

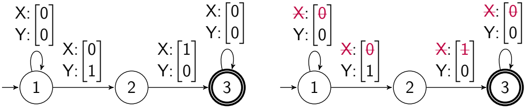
- Remove the x track (project on the y track).



A $\qquad$ $\rightarrow$ Projection on X

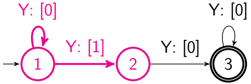
Automata projection, $\exists x$

- Remove the x track (project on the y track).



A $\quad\rightarrow$ Projection on X

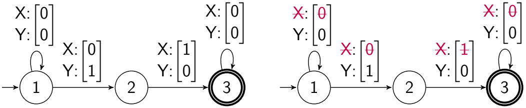
- Careful, does it accept all encodings of sat. assignments?



$10, 010, 0010, \dots$ are accepted,
$1, 01, 001, \dots$ should be accepted too

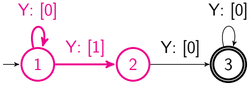
Automata projection, $\exists x$

- Remove the x track (project on the y track).



A $\rightarrow$ Projection on X

- Careful, does it accept all encodings of sat. assignments?

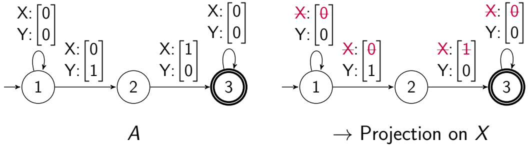


$10, 010, 0010, \ldots$ are accepted,
$1, 01, 001, \ldots$ should be accepted too

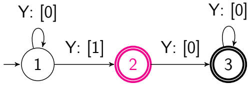
- Saturation acceptance: everything reaching final state by zero vectors becomes also accepting.

Automata projection, $\exists x$

- Remove the x track (project on the y track).



- Careful, does it accept all encodings of sat. assignments?



$10, 010, 0010, \ldots$ are accepted,
$1, 01, 001, \ldots$ should be accepted too

- Saturation acceptance: everything reaching final state by zero vectors becomes also accepting.

Part III

Automata for Atomic Presburger Predicates

Atomic predicates

- Assume that atomic predicates were transformed into the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

where $a_1, \ldots, a_n, b \in \mathbb{Z}$.

Atomic predicates

- Assume that atomic predicates were transformed into the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

  where $a_1, \ldots, a_n, b \in \mathbb{Z}$.
- We write

$$\bar{a} \cdot \bar{x} = b$$

  where $\bar{a} = (a_1, \ldots, a_n)$, $\bar{x} = (x_1, \ldots, x_n)$, and $\bar{a} \cdot \bar{x}$ denotes the scalar product.

States of atomic predicates

Example: $2x - y = 2$

States of atomic predicates

Example: $2x - y = 2$

Idea:

- The states of the automaton are numbers $q \in \mathbb{Z}$.
- From q will be read those assignment to x and y under which $2x - y$ equals q.

States of atomic predicates

Example: $2x - y = 2$

Idea:

- The states of the automaton are numbers $q \in \mathbb{Z}$.
- From q will be read those assignment to x and y under which $2x - y$ equals q.

From state $q \in \mathbb{Z}$ are read encodings of $\bar{c} \in \mathbb{N}^n$ such that $\bar{a} \cdot \bar{c} = q$.

Initial states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- The whole assignment is read from the initial state.

Initial states of atomic predicates $\quad 2x - y = 2$

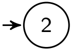
Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- The whole assignment is read from the initial state.
- So 2 must be initial.

Initial states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

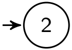
- The whole assignment is read from the initial state.
- So 2 must be initial.



Initial states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- The whole assignment is read from the initial state.
- So 2 must be initial.

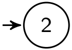


2 is the only initial state

Transitions of atomic predicates $\quad 2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

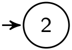
- If $q \xrightarrow{\zeta} q'$ then:



Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

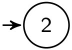
- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'



Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

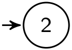
- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q



Transitions of atomic predicates $\quad 2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

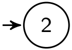
- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$



Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

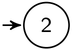
- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$



Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

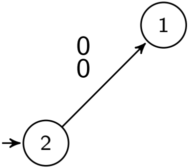


$$\delta(q,\zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Transitions of atomic predicates $\quad 2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

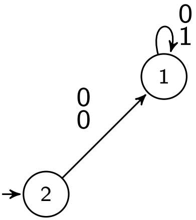


$$\delta(q, \zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Transitions of atomic predicates $\quad 2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

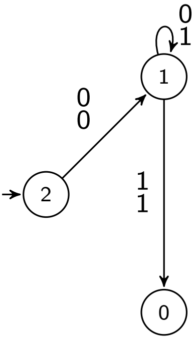


$$\delta(q, \zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

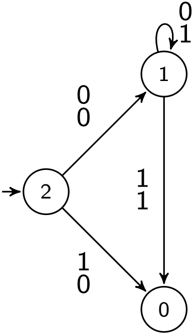


$$\delta(q,\zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

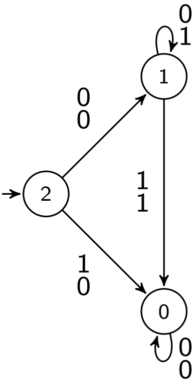


$$\delta(q, \zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

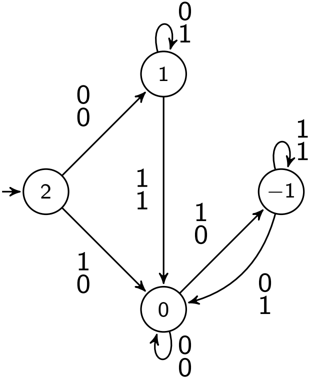


$$\delta(q, \zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Transitions of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- If $q \xrightarrow{\zeta} q'$ then:
- $\bar{c}'$ is read from q'
- iff $2\bar{c}' + \zeta$ read from q
- iff $\bar{a} \cdot (2\bar{c}' + \zeta) = q$
- iff $\bar{a} \cdot \bar{c}' = \frac{1}{2}(q - \bar{a} \cdot \zeta)$

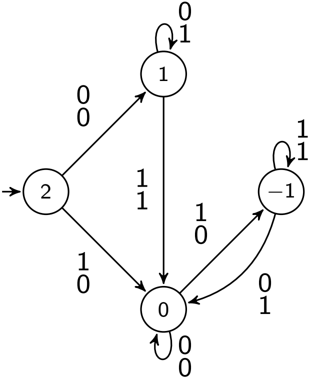


$$\delta(q,\zeta) = \begin{cases} \{\frac{1}{2}(q - \bar{a} \cdot \zeta)\} & \text{if } q - \bar{a} \cdot \zeta \text{ is even} \\ \emptyset & \textit{otherwise} \end{cases}$$

Accepting states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

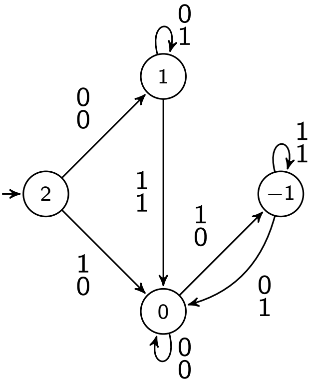
- Accepting are those where nothing (i.e. ϵ) needs to be read to satisfy $\bar{a} \cdot \bar{c} = q$.



Accepting states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

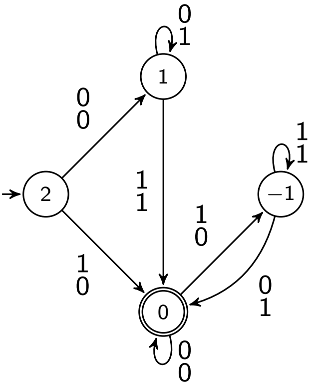
- Accepting are those where nothing (i.e. ϵ) needs to be read to satisfy $\bar{a} \cdot \bar{c} = q$.
- This is only 0.



Accepting states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

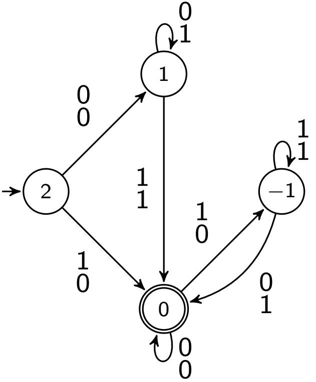
- Accepting are those where nothing (i.e. ϵ) needs to be read to satisfy $\bar{a} \cdot \bar{c} = q$.
- This is only 0.



Accepting states of atomic predicates $2x - y = 2$

Reminder:
from q read $\bar{c}$ s.t. $\bar{a} \cdot \bar{c} = q$

- Accepting are those where nothing (i.e. ϵ) needs to be read to satisfy $\bar{a} \cdot \bar{c} = q$.
- This is only 0.



0 is the only accepting state

Atomic predicates: algorithm

EqtoDFA(φ)
Input: Equation $\varphi = \; a \cdot x = b$
Output: DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L(\varphi)$ (without trap state)

$Q, \delta, F \leftarrow \emptyset$; $q_0 \leftarrow s_b$
$W \leftarrow \{s_b\}$
while $W \neq \emptyset$ **do**
    pick s_k **from** W
    add s_k **to** Q
    if $k = 0$ **then add** s_k **to** F
    for all $\zeta \in \{0, 1\}^n$ **do**
        if $(k - a \cdot \zeta)$ is even **then**
            $j \leftarrow \frac{1}{2}(k - a \cdot \zeta)$
            if $s_j \notin Q$ **then add** s_j **to** W
            add (s_k, ζ, s_j) **to** δ

Remarks

- We have seen a procedure quite different from the ones based on quantifier elimination.
- It can be optimized, extended to $\mathbb{Z}$.
- It shows how diverse solutions of a problem can be,
- and a surprising connection between arithmetic and automata.
- Integer/Presburger arithmetic are somewhat “regular”.

Part IV

Weak Monadic Second Order Logic of One Successor (WS1S)

WS1S

- Minimalistic syntax

$$\varphi \to X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.

WS1S

- Minimalistic syntax

$$\varphi \to X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y.Y \subseteq X \to Y = X$

WS1S

- Minimalistic syntax

$$\varphi \rightarrow X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y. Y \subseteq X \rightarrow Y = X$
- $sing(X) : X \neq \emptyset \wedge (Y \subseteq X \rightarrow (Y = \emptyset \vee X = Y))$

WS1S

- Minimalistic syntax

$$\varphi \rightarrow X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y.Y \subseteq X \rightarrow Y = X$
- $sing(X) : X \neq \emptyset \wedge (Y \subseteq X \rightarrow (Y = \emptyset \vee X = Y))$
  - gives us first order vars.: $\exists x$, $x \in Y$, $x < y$, $x = y$ ...

WS1S

- Minimalistic syntax

$$\varphi \to X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y.Y \subseteq X \to Y = X$
- $sing(X) : X \neq \emptyset \wedge (Y \subseteq X \to (Y = \emptyset \vee X = Y))$
  - gives us first order vars.: $\exists x$, $x \in Y$, $x < y$, $x = y$ ...
- $X = \bigcup_{i=1}^{n} X_i : \bigwedge_{i=1}^{n} X_i \subseteq X \wedge \forall x.(x \in X \to \bigvee_{i=1}^{n} x \in X_i)$

WS1S

- Minimalistic syntax

$$\varphi \rightarrow X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y. Y \subseteq X \rightarrow Y = X$
- $sing(X) : X \neq \emptyset \wedge (Y \subseteq X \rightarrow (Y = \emptyset \vee X = Y))$
  - gives us first order vars.: $\exists x$, $x \in Y$, $x < y$, $x = y$ ...
- $X = \bigcup_{i=1}^{n} X_i : \bigwedge_{i=1}^{n} X_i \subseteq X \wedge \forall x.(x \in X \rightarrow \bigvee_{i=1}^{n} x \in X_i)$
- $Partition(X, X_1, \ldots, X_n)$:
  $X = \bigcup_{i=1}^{n} X_i \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} X_i \cap X_j = \emptyset$

WS1S

- Minimalistic syntax

$$\varphi \to X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y.Y \subseteq X \to Y = X$
- $sing(X) : X \neq \emptyset \wedge (Y \subseteq X \to (Y = \emptyset \vee X = Y))$
  - gives us first order vars.: $\exists x$, $x \in Y$, $x < y$, $x = y$ ...
- $X = \bigcup_{i=1}^{n} X_i : \bigwedge_{i=1}^{n} X_i \subseteq X \wedge \forall x.(x \in X \to \bigvee_{i=1}^{n} x \in X_i)$
- $Partition(X, X_1, \ldots, X_n)$:
  $X = \bigcup_{i=1}^{n} X_i \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} X_i \cap X_j = \emptyset$
- properties of linked data structures: transitive closure of a relation, a graph does not contain cycles, x is reachable from y, ...

WS1S

- Minimalistic syntax

$$\varphi \to X \subseteq X \mid succ(X) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

- interpreted over finite subsets of $\mathbb{N}$.
- $X = \emptyset : \forall Y. Y \subseteq X \to Y = X$
- $sing(X) : X \neq \emptyset \wedge (Y \subseteq X \to (Y = \emptyset \vee X = Y))$
  - gives us first order vars.: $\exists x$, $x \in Y$, $x < y$, $x = y$ ...
- $X = \bigcup_{i=1}^{n} X_i : \bigwedge_{i=1}^{n} X_i \subseteq X \wedge \forall x.(x \in X \to \bigvee_{i=1}^{n} x \in X_i)$
- $Partition(X, X_1, \ldots, X_n)$:
  $X = \bigcup_{i=1}^{n} X_i \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} X_i \cap X_j = \emptyset$
- properties of linked data structures: transitive closure of a relation, a graph does not contain cycles, x is reachable from y, ...
- Also Presburger arithmetic!

The name

Weak monadic second order logic of one successor.

The name

Weak monadic second order logic of one successor.

- **second order** = can quantify over sets

The name

Weak monadic second order logic of one successor.

- **second order** = can quantify over sets
- **weak** = the sets can be finite only

The name

Weak monadic second order logic of one successor.

- **second order** = can quantify over sets
- **weak** = the sets can be finite only
- **monadic** = only sets numbers, not relations

The name

Weak monadic second order logic of one successor.

- **second order** = can quantify over sets
- **weak** = the sets can be finite only
- **monadic** = only sets numbers, not relations
- **one successor** = the *succ* function

Why

- Looks different, but is surprisingly close to Automata, Presburger, regularity.
- It is The basic automata logic. Starting point for many other interesting automata-related logics.
- Exciting research!

Assignments as words

- $X \subseteq \mathbb{N}$ is encoded as a binary vector

$$\{1, 3, 5\} \ldots w = 010101.$$

Assignments as words

- $X \subseteq \mathbb{N}$ is encoded as a binary vector

$$\{1,3,5\} \ldots w = 010101.$$

plus all in $w0^*$

Assignments as words

- $X \subseteq \mathbb{N}$ is encoded as a binary vector

$$\{1,3,5\} \ldots w = 010101.$$

  plus all in $w0^*$

- An assignment, n-tuple of sets, becomes a word over $\{0,1\}^n$.
  $X = \{1,3,4\}$, $Y = \{2\}$, $Z = \emptyset$

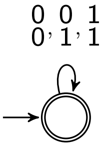
$$\begin{matrix}01011\\00100\\00000\end{matrix}, \begin{matrix}01011\,0\\00100\,0\\00000\,0\end{matrix}, \begin{matrix}01011\,00\\00100\,00\\00000\,00\end{matrix}, \begin{matrix}01011\,000\\00100\,000\\00000\,000\end{matrix}, \cdots$$

WS1S to automata

- automata for $X \subseteq Y$ and $Y = succ(X)$

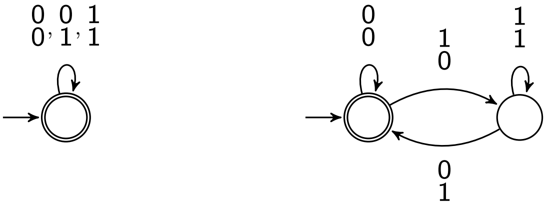
WS1S to automata

- automata for $X \subseteq Y$ and $Y = succ(X)$



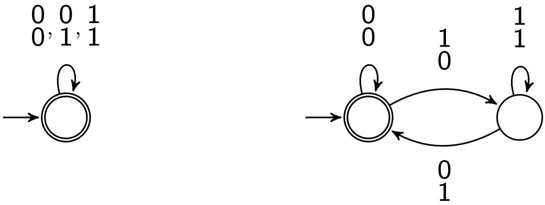
WS1S to automata

- automata for $X \subseteq Y$ and $Y = succ(X)$



WS1S to automata

- automata for $X \subseteq Y$ and $Y = succ(X)$



- and everything else is the same as for Presburger!

Remarks

- WS1S can encode Presburger, not the other way around. Presburger would need the bit-subset operator.

Remarks

- WS1S can encode Presburger, not the other way around. Presburger would need the bit-subset operator.
- Other similar logics
  - S1S for reasoning about arbitrary sets (automata over infinite words)
  - (W)SkS allows many successors — reasoning about tress and general graphs (tree automata)

Remarks

- WS1S can encode Presburger, not the other way around. Presburger would need the bit-subset operator.
- Other similar logics
  - S1S for reasoning about arbitrary sets (automata over infinite words)
  - (W)SkS allows many successors — reasoning about tress and general graphs (tree automata)
- Regularity for words: WS1S = automata [Büchi 1960] = regular expressions = Presburger with bit-subset

Remarks

- WS1S can encode Presburger, not the other way around. Presburger would need the bit-subset operator.
- Other similar logics
  - S1S for reasoning about arbitrary sets (automata over infinite words)
  - (W)SkS allows many successors — reasoning about tress and general graphs (tree automata)
- Regularity for words: WS1S $=$ automata [Büchi 1960] $=$ regular expressions $=$ Presburger with bit-subset
- Complexity
  - Presburger with automata (a bit different algo.): $\mathcal{O}(2^{2^{2^n}})$
  - WS1S: non-elementary complexity $\mathcal{O}(\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}_{\text{alternations}})$