What is in the course?

Analysis of Counter Systems
From Reachability to Temporal Logics

• Part I
  • Lecture 1: Introduction to counter systems
  • Lecture 2: Classes of semilinear reachability sets
  • Lecture 3: Vector addition systems with states

• Part II
  • Lecture 4: Linear-time temporal logics
  • Lecture 5: Model-checking counter systems
  • Lecture 6: Data logics and counter systems

but also Presburger arithmetic, undecidability, computational complexity, etc.
What can you expect to learn?

- Presentation of numerous **classes of counter systems**.

- **Proof techniques** to decide reachability problems for infinite-state systems.

- **Temporal reasoning** on transition systems.
Background

1 Necessary background

- Basics of first-order logic and temporal logics.
- Basics of finite-state automata and formal languages.
- Basics of complexity theory, decidability.

2 Optional background

- Temporal logic LTL and automata-based approach.
- Petri nets.
- Familiarity with complexity classes NP, PSPACE, EXPSPACE etc.
Course material

- Main material can be found in ESSLLI 2010 lecture notes:

  www.lsv.ens-cachan.fr/~demri/esslli10-course.html

- Slides shall be available on the dedicated web page:

  www.lsv.ens-cachan.fr/~demri/UBAUNC10-course.php
Formal Verification
Verification at the heart of computer science

- Digital systems are everywhere. Desktops, embedded systems, cellular phones, etc.
Verification at the heart of computer science

• Digital systems are everywhere.
  Desktops, embedded systems, cellular phones, etc.

• Needs for verifying functional/security properties:
  • Hardware components
  • Software (programs, communication protocols, web applications, . . . )
Verification at the heart of computer science

- Digital systems are everywhere. Desktops, embedded systems, cellular phones, etc.

- Needs for verifying functional/security properties:
  - Hardware components
  - Software (programs, communication protocols, web applications, ...)

Formal verification is a process in which mathematical techniques are used to guarantee the correctness of a design with respect to some specified behavior.

[Halpern et al., BSL 01]
From systems to models

• Systems are modelled as abstract operational models (counter systems, timed automata, etc.).
From systems to models

- Systems are modelled as abstract operational models (counter systems, timed automata, etc.).
Verification as a logical problem

- Properties are represented by logical formula.
  “The system $S$ never reaches a bad state” $\rightarrow \Box G \neg \text{bad}$.
Verification as a logical problem

- Properties are represented by logical formula. “The system $S$ never reaches a bad state” $\rightarrow \forall G \neg \text{bad}$. 

- Logical problems involve abstract models and formulae.
Verification as a logical problem

- Properties are represented by logical formula. “The system $S$ never reaches a bad state” $\rightarrow \forall G \neg \text{bad}$.

- Logical problems involve abstract models and formulae.

- Development of procedures to solve these problems. automata, analytic proof systems, ad-hoc methods . . .
Verification as a logical problem

- Properties are represented by logical formula. “The system $S$ never reaches a bad state” → $\forall G \neg \text{bad}$.

- Logical problems involve abstract models and formulae.

- Development of procedures to solve these problems. automata, analytic proof systems, ad-hoc methods . . .

- Ultimate goal: automatic verification.
Verification as a logical problem

- Properties are represented by logical formula. “The system $S$ never reaches a bad state” $\rightarrow \forall G \neg \text{bad}$.

- Logical problems involve abstract models and formulae.

- Development of procedures to solve these problems. automata, analytic proof systems, ad-hoc methods . . .

- Ultimate goal: automatic verification.

- There are theoretical limits for this entreprise.
  - The halting problem for Turing machines is undecidable. [Turing, 37]
Verification as a logical problem

• Properties are represented by logical formula. “The system $S$ never reaches a bad state” $\rightarrow A \Diamond \neg \text{bad}$.

• Logical problems involve abstract models and formulae.

• Development of procedures to solve these problems. automata, analytic proof systems, ad-hoc methods . . .

• Ultimate goal: automatic verification.

• There are theoretical limits for this enterprise.
  • The halting problem for Turing machines is undecidable. [Turing, 37]
  • The set of valid first-order formulae is undecidable. [Church, JSL 36]
Methodology

- System, property $\mapsto$ model, logical formula.
Methodology

• System, property $\rightarrow$ model, logical formula.

• Logical problems:
  • Decision problems (model-checking, validity, . . . )
  • Search problems (controller synthesis, query checking, . . . )
Methodology

- System, property $\mapsto$ model, logical formula.

- Logical problems:
  - Decision problems (model-checking, validity, ...)
  - Search problems (controller synthesis, query checking, ...)

- Analysis of the computational resources to solve the problems
  - Decision procedures vs. undecidability.
  - Complexity in time or memory space.
  - Worst-case complexity, parameterized complexity, etc.
Methodology

- System, property $\mapsto$ model, logical formula.

- Logical problems:
  - Decision problems (model-checking, validity, . . .)
  - Search problems (controller synthesis, query checking, . . .)

- Analysis of the computational resources to solve the problems
  - Decision procedures vs. undecidability.
  - Complexity in time or memory space.
  - Worst-case complexity, parameterized complexity, etc.

- Classification
  - Generalizing the models or logics (e.g., Extended TL).
  - Fragments with better computational properties (e.g., FO2).
Formal verification and temporal logics

- Aspects of temporality in computer science
  - Specification and verification of concurrent/reactive systems.
  - Real-time processes and systems.
  - Temporal databases.

- Logics as formal specification languages
  - To define mathematically the correctness of systems.
  - To express properties without ambiguities.
  - To make formal proofs and develop generic methods.
Model-checking and temporal logic

- Temporal logic for specifying behaviors of reactive systems [Pnueli, FOCS 77].
Model-checking and temporal logic

- Temporal logic for specifying behaviors of reactive systems [Pnueli, FOCS 77].

- Model-checking approach:
  - Computer system is modelled as a graph/model $M$.
  - Specification is a temporal logic formula $\varphi$.
  - Check whether $M$ satisfies $\varphi$ ($M \models \varphi$).

(Turing award recipients’07: Clarke & Emerson & Sifakis)
Model-checking and temporal logic

- Temporal logic for specifying behaviors of reactive systems [Pnueli, FOCS 77].

- Model-checking approach:
  - Computer system is modelled as a graph/model $\mathcal{M}$.
  - Specification is a temporal logic formula $\varphi$.
  - Check whether $\mathcal{M}$ satisfies $\varphi$ ($\mathcal{M} \models \varphi$).

  (Turing award recipients’07: Clarke & Emerson & Sifakis)

- Automata-based approach
  (Gödel prize 2000) [Vardi & Wolper, IC 94]
Model-checking and temporal logic

• Temporal logic for specifying behaviors of reactive systems [Pnueli, FOCS 77].

• Model-checking approach:
  • Computer system is modelled as a graph/model $M$.
  • Specification is a temporal logic formula $\varphi$.
  • Check whether $M$ satisfies $\varphi$ ($M \models \varphi$).

  (Turing award recipients’07: Clarke & Emerson & Sifakis)

• Automata-based approach
  (Gödel prize 2000) [Vardi & Wolper, IC 94]

• Early work on logic and automata. [Büchi, 62]
In this Course: Focus on Counter Systems
Ubiquity of counter systems

- Counter system: finite-state automaton with counters interpreted by non-negative integers.
- Techniques for model-checking infinite-state systems are required for formal verification.
Ubiquity of counter systems

- Counter system: finite-state automaton with counters interpreted by non-negative integers.

- Techniques for model-checking infinite-state systems are required for formal verification.

- Many applications:
  - Broadcast protocols, Petri nets, . . .
  - Programs with pointer variables.  [Bouajjani et al., CAV’06]
  - Replicated finite-state programs.  [Kaiser & Kroening & Wahl, CAV’10]
  - Relationships with data logics.  [Bojańczyk et al., LICS’06]
  - . . .
Ubiquity of counter systems

- Counter system: finite-state automaton with counters interpreted by non-negative integers.

- Techniques for model-checking infinite-state systems are required for formal verification.

- Many applications:
  - Broadcast protocols, Petri nets, . . .
  - Programs with pointer variables. [Bouajjani et al., CAV’06]
  - Replicated finite-state programs. [Kaiser & Kroening & Wahl, CAV’10]
  - Relationships with data logics. [Bojańczyk et al., LICS’06]
  - . . .

- But, counter systems can simulate Turing machines.

- Checking safety or liveness properties for counter systems are undecidable problems.
Taming counter systems

- Design of subclasses with decidable reachability problems
  - Vector addition systems (\(\approx\) Petri nets).
    [Mayr, STOC 81; Kosaraju, STOC’82]
  - Flat relational counter systems. [Comon & Jurski, CAV’98]
  - Reversal-bounded counter automata. [Ibarra, JACM 78]
  - Flat affine counter systems.
    [Boigelot, PhD 98; Finkel & Leroux, FSTTCS’02]
- . . .
Taming counter systems

- Design of subclasses with decidable reachability problems
  - Vector addition systems (≈ Petri nets).
    [Mayr, STOC 81; Kosaraju, STOC’82]
  - Flat relational counter systems. [Comon & Jurski, CAV’98]
  - Reversal-bounded counter automata. [Ibarra, JACM 78]
  - Flat affine counter systems.
    [Boigelot, PhD 98; Finkel & Leroux, FSTTCS’02]
  - 
  - Decision procedures
    - Translation into Presburger arithmetic (see 2nd lecture).
    - Direct analysis on runs (see 3rd lecture). [Rackoff, TCS 78]
    - Approximating reachability sets. [Karp & Miller, JCSS 69]
    - Well-structured transition systems.
      [Finkel & Schnoebelen, TCS 01]
  - Approximating reachability sets. [Karp & Miller, JCSS 69]
Taming counter systems

- Design of subclasses with decidable reachability problems
  - Vector addition systems (≈ Petri nets).
    [Mayr, STOC 81; Kosaraju, STOC’82]
  - Flat relational counter systems. [Comon & Jurski, CAV’98]
  - Reversal-bounded counter automata. [Ibarra, JACM 78]
  - Flat affine counter systems.
    [Boigelot, PhD 98; Finkel & Leroux, FSTTCS’02]
  - ...

- Decision procedures
  - Translation into Presburger arithmetic (see 2nd lecture).
  - Direct analysis on runs (see 3rd lecture). [Rackoff, TCS 78]
  - Approximating reachability sets. [Karp & Miller, JCSS 69]
  - Well-structured transition systems.
    [Finkel & Schnoebelen, TCS 01]

- Tools: MONA, FAST, LASH, TREX, TAPAS, SMT solvers etc.
Toy Example: Pay Phone Controller
\(x_1 = x_2 = 0, \text{lift?}\)

\(x_1 = x_2, x_1' = x_2' = 0\)

\(x_1 + +, \text{coin?} \quad \text{if } x_1 + +, \text{coin?} \quad \text{then } \)

\(x_2 < x_1, \text{signal?}, x_2 + +\)

\(x_2 \leq x_1\)

\(x_2' \leq x_1, x_2 + +, \text{coin!}\)

- \(x_1\): number of coins which have been inserted.
- \(x_2\): number of time units spent for communication.
- \(x_1'\) [resp. \(x_2'\)] is the next value of \(x_1\) [resp. \(x_2\)].
- \(x_1 + +\) is a shortcut for \(x_1' = x_1 + 1 \land x_2' = x_2\).
How to read the figure

- $q_1$ is the initial state and the final state.

- $x_1$ and $x_2$ can only take non-negative values.

- The controller interacts with the environment including the phone box. It can receive or send messages.

- Message ’coin?’: the controller receives the information that a coin has been inserted.

- Message ’coin!’: the controller sends the information that a coin has been released.
Underlying infinite transition system

- Configuration: description of the current state of the system.
Underlying infinite transition system

- Configuration: description of the current state of the system.

- A configuration is a triple \((q, n_1, n_2)\) where \(q\) is a control state and \(n_1\) [resp. \(n_2\)] is the value of \(x_1\) [resp. \(x_2\)].

- Because of the presence of messages, queues for messages should be added (omitted here).
Underlying infinite transition system

- Configuration: description of the current state of the system.

- A configuration is a triple \((q, n_1, n_2)\) where \(q\) is a control state and \(n_1\) [resp. \(n_2\)] is the value of \(x_1\) [resp. \(x_2\)].

- Because of the presence of messages, queues for messages should be added (omitted here).

- An execution is a (possibly infinite) sequence of configurations constrained by the system.

- Unbounded insertion of coins:
  \[
  (q_1, 0, 0), (q_2, 0, 0), (q_2, 1, 0), (q_2, 2, 0), (q_2, 3, 0), \ldots
  \]

- This system is a finite and concise representation of an infinite labeled transition system.
Which properties hold true?

- Total communication time is never greater than the number of inserted coins:
  \[ A \ G \neg(x_2 > x_1). \]

- For all infinite executions, the number of coins is infinitely often equal to zero:
  \[ A \ G \ F (x_1 = 0). \]

- There is an execution of the controller such that the total communication time is always equal to zero:
  \[ E \ G (x_2 = 0). \]

- Whenever the communication is over, eventually the system can reach the initial configuration:
  \[ A \ G (q_5 \Rightarrow F q_1). \]

- Whenever the control state \( q_1 \) is reached, \( x_1 = x_2 = 0 \) and conversely:
  \[ A \ G (q_1 \Leftrightarrow (x_1 = 0 \land x_2 = 0)). \]
A Fundamental Model: Minsky Machines
Deterministic Minsky machines

- A counter stores a single natural number.

- A Minsky machine can be viewed as a finite-state machine with two counters.

- Operations on counters:
  - Check whether the counter is zero.
  - Increment the counter by one.
  - Decrement the counter by one if nonzero.
2-counter Minsky machines

- Set of $n$ instructions.

- The $l$th instruction has one of the forms below ($i \in \{1, 2\}$, $l' \in \{1, \ldots, n\}$):
  
  I: $C_i := C_i + 1; \text{ goto } l'$
  
  I: if $C_i = 0$ then goto $l'$ else $C_i := C_i - 1; \text{ goto } l''$.

- Configurations are elements of $\{1, \ldots, n\} \times \mathbb{N} \times \mathbb{N}$.

- Initial configuration: $(1, 0, 0)$. 
A computation is a sequence of configurations starting from the initial configuration and such that two successive configurations respect the instructions.

The Minsky machine

1: $C_1 := C_1 + 1; \text{ goto } 2$
2: $C_2 := C_2 + 1; \text{ goto } 1$

has unique computation

$(1, 0, 0) \rightarrow (2, 1, 0) \rightarrow (1, 1, 1) \rightarrow (2, 2, 1) \rightarrow (1, 2, 2) \rightarrow (2, 3, 2) \ldots$
Halting problem

- Halting problem:
  - **input:** a 2-counter Minsky machine $M$;
  - **question:** is there a finite computation that ends with location equal to $n$?
  
  ($n$ may also be a special instruction that halts the machine)

- **Theorem:** The halting problem is undecidable.

  [Minsky, 67]
Halting problem

- Halting problem:
  - **input**: a 2-counter Minsky machine $M$;
  - **question**: is there a finite computation that ends with location equal to $n$?
  
  ($n$ may also be a special instruction that halts the machine)

- **Theorem**: The halting problem is undecidable.
  
  [Minsky, 67]

- Minsky machines are Turing-complete (see next slide).
Turing machines

- Nondeterministic Turing machine $M = (Q, q_0, \Sigma, \delta, q_a)$:
  - $Q$: set of control states.
  - $q_0$: initial state; $q_a$: accepting state.
  - $\Sigma$: tape symbols (including a blank symbol or an end symbol).
  - Transition relation $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q \times \{−1, 0, 1\} \times \Sigma)$.

- We can assume that the Turing machine starts with an “empty” tape.

- The halting problem for Turing machines is undecidable [Turing, 1936].
Simulating a Turing machine (ideas only)

- A Turing machine can be simulated by two stacks (the tape is cut in half).
  - E.g., moving the head left or right is equivalent to popping a bit from one stack and pushing it onto the other.
Simulating a Turing machine (ideas only)

- A Turing machine can be simulated by two stacks (the tape is cut in half).
  - E.g., moving the head left or right is equivalent to popping a bit from one stack and pushing it onto the other.

- A stack over a binary alphabet can be simulated by two counters. One counter contains the binary representation of the bits on the stack.
  - E.g., pushing a one is equivalent to doubling and adding 1, assuming that in the binary representation the least significant bit is on the top.
Simulating a Turing machine (ideas only)

- A Turing machine can be simulated by two stacks (the tape is cut in half).
  - E.g., moving the head left or right is equivalent to popping a bit from one stack and pushing it onto the other.

- A stack over a binary alphabet can be simulated by two counters. One counter contains the binary representation of the bits on the stack.
  - E.g., pushing a one is equivalent to doubling and adding 1, assuming that in the binary representation the least significant bit is on the top.

- Four counters can be simulated by two counters.
  - Counter values \((a, b, c, d)\) encoded by value \(2^a3^b5^c7^d\).
  - E.g., checking the third counter is zero is equivalent to dividing by 5 and see what the remainder is. The second counter is auxiliary.
Non-deterministic Minsky machines

- Nondeterministic choice after incrementation and decrementation.

- Instructions are of the forms below:
  \[ i: \quad C_i := C_i + 1; \text{ goto } l' \text{ or goto } l'' \]
  \[ i: \quad \text{ if } C_i = 0 \text{ then goto } l' \text{ else } C_i := C_i - 1; \text{ goto } l'' \text{ or goto } l_1'' . \]
Non-deterministic Minsky machines

- Nondeterministic choice after incrementation and decrementation.

- Instructions are of the forms below:
  \[ l: \quad C_i := C_i + 1; \text{ goto } l' \text{ or goto } l'' \]
  \[ l: \quad \text{if } C_i = 0 \text{ then goto } l' \text{ else } C_i := C_i - 1; \text{ goto } l'' \text{ or goto } l''' \]

- Recurrence problem:
  \textbf{input:} a NDM Minsky machine \( M \);
  \textbf{question:} is there an infinite computation with instruction 1 occurring infinitely often?

- The recurrence problem is \( \Sigma_1 \)-complete, i.e. highly undecidable.

[Alur & Henzinger, JACM 94]
Minsky machines: an assembly language?

- Minsky machines have a strong computational power.
- But, it is unlikely that one may wish to solve decision problems by programming Minsky machines.
Minsky machines: an assembly language?

- Minsky machines have a strong computational power.
- But, it is unlikely that one may wish to solve decision problems by programming Minsky machines.
- Problems on Minsky machines are easily undecidable.
Minsky machines: an assembly language?

- Minsky machines have a strong computational power.
- But, it is unlikely that one may wish to solve decision problems by programming Minsky machines.
- Problems on Minsky machines are easily undecidable.
- Counter systems will allow more flexibility and admit a richer set of instructions.
- ...but, first we need to present Presburger arithmetic.
Presburger Arithmetic
A fundamental decidable theory

- First-order theory of $(\mathbb{N}, +)$ introduced by Mojcesz Presburger (1929).

- Instrumental to constrain counter values in counter systems.

- Formulae are viewed as symbolic representations for (infinite) sets of tuples of natural numbers.

- A first-order theory with many interesting properties:
  - Decidability (by contrast to first-order theory of $(\mathbb{N}, +, \times)$).
  - Sets definable in Presburger arithmetic are precisely semilinear sets (see next slides).

- Formalism also used to express constraints on graphs, on number of events, etc.

  See e.g., [Seidl & Schwentick & Muscholl, chapter 07]
Presburger arithmetic [Presburger, 29]

- “First-order theory of $\langle \mathbb{N}, + \rangle$” (no multiplication).

- Terms: $t ::= 0 \mid 1 \mid x \mid t + t$.

- $2x + 3$ is a shortcut for $x + x + 1 + 1 + 1$. 
Presburger arithmetic [Presburger, 29]

• “First-order theory of \((\mathbb{N}, +)\)” (no multiplication).

• Terms: \( t ::= 0 \mid 1 \mid x \mid t + t. \)

• \(2x + 3\) is a shortcut for \(x + x + 1 + 1 + 1.\)

• Presburger formulae \((k \geq 2)\)

\[ \varphi ::= t \equiv_k t \mid t < t \mid \neg \varphi \mid \varphi \land \varphi \mid \exists x \, \varphi \mid \forall x \, \varphi \]

• Valuation \(v : \text{VAR} \rightarrow \mathbb{N} + \) extension to all terms with

\[ v(0) = 0 \quad v(1) = 1 \quad v(t + t') = v(t) + v(t') \]

• Oddness: \(\exists y \ x = y + y + 1.\)

( with \( “t = t’” \overset{\text{def}}{=} \neg(t < t' \lor t' < t)”) \)
Semantics

- \( \mathbf{v} \models t \equiv^k t' \ \overset{\text{def}}{\iff} \) there is \( m \in \mathbb{Z} \) such that \( km + \mathbf{v}(t) = \mathbf{v}(t') \),

- \( \mathbf{v} \models t < t' \ \overset{\text{def}}{\iff} \) \( \mathbf{v}(t) < \mathbf{v}(t') \),

- \( \mathbf{v} \models \neg \varphi \ \overset{\text{def}}{\iff} \) \( \mathbf{v} \not\models \varphi \),

- \( \mathbf{v} \models \varphi \land \varphi' \ \overset{\text{def}}{\iff} \) \( \mathbf{v} \models \varphi \) and \( \mathbf{v} \models \varphi' \),

- \( \mathbf{v} \models \exists x \ \varphi \ \overset{\text{def}}{\iff} \) there is \( n \in \mathbb{N} \) such that \( \mathbf{v}[x \mapsto n] \models \varphi \) where \( \mathbf{v}[x \mapsto n] \) is equal to \( \mathbf{v} \) except that \( x \) is mapped to \( n \),

- \( \mathbf{v} \models \forall x \ \varphi \ \overset{\text{def}}{\iff} \) for every \( n \in \mathbb{N} \), we have \( \mathbf{v}[x \mapsto n] \models \varphi \).
Semantics

- \( v \models t \equiv_k t' \stackrel{\text{def}}{\iff} \text{there is } m \in \mathbb{Z} \text{ such that } km + v(t) = v(t'), \)
- \( v \models t < t' \stackrel{\text{def}}{\iff} v(t) < v(t'), \)
- \( v \models \neg \varphi \stackrel{\text{def}}{\iff} v \not\models \varphi, \)
- \( v \models \varphi \land \varphi' \stackrel{\text{def}}{\iff} v \models \varphi \text{ and } v \models \varphi', \)
- \( v \models \exists x \varphi \stackrel{\text{def}}{\iff} \text{there is } n \in \mathbb{N} \text{ such that } v[x \rightarrow n] \models \varphi \text{ where } v[x \rightarrow n] \text{ is equal to } v \text{ except that } x \text{ is mapped to } n, \)
- \( v \models \forall x \varphi \stackrel{\text{def}}{\iff} \text{for every } n \in \mathbb{N}, \text{ we have } v[x \rightarrow n] \models \varphi. \)

\( t \equiv_k t' \) is equivalent to \( \exists x \ (t = kx + t') \lor (t' = kx + t) \).
Defining sets of tuples

- Formula $\varphi(x_1, \ldots, x_n)$ with $n$ free variables:

  \[
  \text{REL}(\varphi(x_1, \ldots, x_n)) \overset{\text{def}}{=} \{(v(x_1), \ldots, v(x_n)) \in \mathbb{N}^n : v \models \varphi\}.
  \]

- $\varphi$ is satisfiable $\iff$ there is $v$ such that $v \models \varphi$.

- $\varphi$ is valid $\iff$ for all $v$, we have $v \models \varphi$.

- If $\varphi$ has no free variable, then satisfiability is equivalent to validity.

- $\varphi(x_1, \ldots, x_n)$ is valid iff $\forall x_1, \ldots, x_n \varphi(x_1, \ldots, x_n)$ is satisfiable/valid.
Decidability and quantifier elimination

- **Theorem:** The satisfiability problem for Presburger arithmetic is decidable. [Presburger, 29]

- Every Presburger formula is **effectively equivalent** to a Presburger formula without first-order quantification. [Presburger, 29]

  (periodicity atomic formulae are needed here)

- Satisfiability problem for quantifier-free formulae is NP-complete. [Papadimitriou, JACM 81]

  See also [Borosh & Treybig, AMS 76]

- About other first-order theories
  - Skolem arithmetic \((\mathbb{N}, 0, 1, \times)\) is decidable.
  - \((\mathbb{Z}, 0, 1, <, +)\) is decidable.
  - \((\mathbb{N}, 0, 1, \times, +)\) is undecidable.
Semilinear sets

- A linear set $X$ is defined by a basis $\vec{b} \in \mathbb{N}^k$ and a finite set of periods $\{\vec{p}_1, \ldots, \vec{p}_m\}$:

$$X = \{ \vec{b} + \sum_{i=1}^{i=m} n_i \vec{p}_i : n_1, \ldots, n_m \in \mathbb{N} \}$$

- A semilinear set is a finite union of linear sets.

- A linear set:

$$\left\{ \begin{pmatrix} 3 \\ 4 \end{pmatrix} + i \times \begin{pmatrix} 2 \\ 5 \end{pmatrix} + j \times \begin{pmatrix} 4 \\ 7 \end{pmatrix} : i, j \in \mathbb{N} \right\}$$

- Subsets of $\mathbb{N}$ that are not semilinear:

  - $\{2^i : i \in \mathbb{N}\}$.
  - $\{i^2 : i \in \mathbb{N}\}$.  

$X = \{2^i : i \in \mathbb{N}\}$ is not semilinear

- Suppose that $X$ is semilinear.

- Since $X$ is infinite, there are $b \in \mathbb{N}$ and $p_1 > \cdots > p_m > 0$ ($m \geq 1$) such that

  \[ Y = \{b + \sum_{i=1}^{i=m} n_ip_i : n_1, \ldots, n_m \in \mathbb{N}\} \subseteq X \]

- Let $2^\alpha \in Y$ such that $p_1 < 2^\alpha$.

- By definition of $Y$, we have $2^\alpha + p_1 \in Y$.

- However, $2^\alpha < 2^\alpha + p_1 < 2^{\alpha+1}$, which leads to a contradiction.
The fundamental characterization
[Ginsburg & Spanier, PJM 66]

- For every Presburger formula $\varphi$ with $n \geq 1$ free variables, $\text{REL}(\varphi)$ is a semilinear subset of $\mathbb{N}^n$.

- For every semilinear set $X \subseteq \mathbb{N}^n$, there is $\varphi$ such that $X = \text{REL}(\varphi)$. 
The fundamental characterization
[Ginsburg & Spanier, PJM 66]

- For every Presburger formula $\varphi$ with $n \geq 1$ free variables, $\text{REL}(\varphi)$ is a semilinear subset of $\mathbb{N}^n$.

- For every semilinear set $X \subseteq \mathbb{N}^n$, there is $\varphi$ such that $X = \text{REL}(\varphi)$.

- The class of semilinear sets are effectively closed under union, intersection, complementation and projection.

- For instance, $(X_1 = \text{REL}(\varphi_1)$ and $X_2 = \text{REL}(\varphi_2))$ imply $X_1 \cap X_2 = \text{REL}(\varphi_1 \land \varphi_2)$
The fundamental characterization
[Ginsburg & Spanier, PJM 66]

- For every Presburger formula \( \varphi \) with \( n \geq 1 \) free variables, \( \text{REL}(\varphi) \) is a semilinear subset of \( \mathbb{N}^n \).
- For every semilinear set \( X \subseteq \mathbb{N}^n \), there is \( \varphi \) such that \( X = \text{REL}(\varphi) \).
- The class of semilinear sets are effectively closed under union, intersection, complementation and projection.
- For instance, \((X_1 = \text{REL}(\varphi_1) \text{ and } X_2 = \text{REL}(\varphi_2))\) imply \( X_1 \cap X_2 = \text{REL}(\varphi_1 \land \varphi_2) \)
- Presburger formula for

\[
\left\{ \begin{pmatrix} 3 \\ 4 \end{pmatrix} + i \times \begin{pmatrix} 2 \\ 5 \end{pmatrix} + j \times \begin{pmatrix} 4 \\ 7 \end{pmatrix} : i, j \in \mathbb{N} \right\}
\]

\[
\exists \, I, J \ (x_1 = 3 + 2I + 4J \land x_2 = 4 + 5I + 7J)
\]
Parikh image

- $\Sigma = \{a_1, \ldots, a_k\}$ with ordering $a_1 < \cdots < a_k$.

- Parikh image of $u \in \Sigma^*$:

  $$
  \begin{pmatrix}
  n_1 \\
  n_2 \\
  \vdots \\
  n_k
  \end{pmatrix} \in \mathbb{N}^k
  $$

  where each $n_j$ is the number of occurrences of $a_j$ in $u$.

- Parikh image of $a b a a b$ is $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$.

- Definition for Parikh image extends to languages.

- The Parikh image of any context-free language is semilinear. [Parikh, JACM 66]

- Effective computation from pushdown automata.
Counter Systems
Counter systems

- Counter system = finite-state automaton + counters governed by Presburger formulae.

\[ x'_1 = x_1 + 1 \quad x'_2 = x_2 + 1 \quad x'_3 = x_3 + 1 \]

- Labels on transitions are Presburger formulae with
  - \( \vec{x} = x_1, x_2, x_3 \) (current values).
  - \( x' = x'_1, x'_2, x'_3 \) (next values).

\[ x'_1 = x'_2 = x'_3 = 0 \]
A simple counter system

1: \( C_1 := C_1 + 1 \); goto 2
2: \( C_2 := C_2 + 1 \); goto 1

\[
x'_1 = x_1 + 1 \land x'_2 = x_2
\]

\[
x'_2 = x_2 + 1 \land x'_1 = x_1
\]
A formal definition

- Counter system $S = (Q, n, \delta)$ of dimension $n$:
  - $Q$ is a nonempty finite set of control states.
  - $n \geq 1$ is the dimension.
  - $\delta$ is the transition relation: finite set of transitions of the form
    \[ t = (q, \varphi, q') \]
    where $q, q' \in Q$ and $\varphi$ is a Presburger formula with free variables $x_1, \ldots, x_n, x'_1, \ldots, x'_n$.

- Prime variables are intended to be interpreted as the next values of the unprimed variables.
Interpretation: transition system

- Configuration \((q, \vec{y}) \in Q \times \mathbb{N}^n\).

- Let us define the valuation \(v_{\vec{y}, \vec{y}'} \approx v[\vec{x} \leftarrow \vec{y}, \vec{x}' \leftarrow \vec{y}']\). For \(i \in [1, n]\),
  1. \(v_{\vec{y}, \vec{y}'}(x_i) \overset{\text{def}}{=} \vec{y}(i)\),
  2. \(v_{\vec{y}, \vec{y}'}(x'_i) \overset{\text{def}}{=} \vec{y}'(i)\).

- Given \(t = q \xrightarrow{\varphi} q', (q, \vec{y}) \xrightarrow{t} (q', \vec{y}') \overset{\text{def}}{=} v_{\vec{y}, \vec{y}'} \models \varphi\).

- Transition system \(T(S) = (S, \rightarrow)\)
  - \(S = Q \times \mathbb{N}^n\),
  - \((q, \vec{y}) \rightarrow (q', \vec{y}') \overset{\text{def}}{=} \exists t \in \delta \text{ s.t. } (q, \vec{y}) \xrightarrow{t} (q', \vec{y}')\).
  - Reflexive and transitive closure \(\rightarrow^*\).

- Runs as nonempty (possibly infinite) sequences
  \[
  \rho = (q_0, \vec{y}_0) \rightarrow (q_1, \vec{y}_1) \cdots (q_k, \vec{y}_k) \cdots
  \]
Reachability problems

- **Reachability Problem:**
  
  **Input:** counter system $S$, $(q, \vec{x})$ and $(q', \vec{x}')$.
  
  **Question:** is there a finite run with initial configuration $(q, \vec{x})$ and final configuration $(q', \vec{x}')$? (in symbols $(q, \vec{x}) \rightarrow^* (q', \vec{x}')$)
Reachability problems

- **Reachability Problem**:
  
  **Input**: counter system $S$, $(q, \vec{x})$ and $(q', \vec{x}')$.
  
  **Question**: is there a finite run with initial configuration $(q, \vec{x})$ and final configuration $(q', \vec{x}')$?
  
  (in symbols $(q, \vec{x}) \xrightarrow{*} (q', \vec{x}')$?)

- **Control State Reachability Problem**:
  
  **Input**: counter system $S$, $(q, \vec{x})$ and $q'$. 
  
  **Question**: is there a finite run with initial configuration $(q, \vec{x})$ and whose final configuration has control state $q'$?
  
  $(\exists \vec{x}' (q, \vec{x}) \xrightarrow{*} (q', \vec{x}')$?)
Reachability problems

- **Reachability Problem:**
  
  **Input:** counter system $S$, $(q, \vec{x})$ and $(q', \vec{x}')$.
  
  **Question:** is there a finite run with initial configuration $(q, \vec{x})$ and final configuration $(q', \vec{x}')$?
  
  (in symbols $(q, \vec{x}) \xrightarrow{*} (q', \vec{x}')$?)

- **Control State Reachability Problem:**
  
  **Input:** counter system $S$, $(q, \vec{x})$ and $q'$.
  
  **Question:** is there a finite run with initial configuration $(q, \vec{x})$ and whose final configuration has control state $q'$?
  
  $(\exists \vec{x}' (q, \vec{x}) \xrightarrow{*} (q', \vec{x}')$?)

- **Control State Repeated Reachability Problem:**
  
  **Input:** counter system $S$, $(q, \vec{x})$ and $q_f$.
  
  **Question:** is there an infinite run with initial configuration $(q, \vec{x})$ such that the control state $q_f$ is repeated infinitely often?
Variant problems

- **Covering Problem:**
  
  **Input:** counter system $S$, $(q, \vec{x})$ and $(q', \vec{x}')$.
  
  **Question:** is there a finite run with initial configuration $(q, \vec{x})$ and whose final configuration is $(q', \vec{x}'')$ with $\vec{x}' \preceq \vec{x}''$?

  (control state reachability is an instance with $\vec{x}' = \vec{0}$)
Variant problems

- **Covering Problem:**
  
  **Input:** counter system $S$, $(q, \vec{x})$ and $(q', \vec{x}')$.
  
  **Question:** is there a finite run with initial configuration $(q, \vec{x})$ and whose final configuration is $(q', \vec{x}''')$ with $\vec{x}' \preceq \vec{x}'''$?
  
  (control state reachability is an instance with $\vec{x}' = \vec{0}$)

- **Boundedness Problem:**
  
  **Input:** counter system $S$ and $(q, \vec{x})$.
  
  **Question:** is the set $\{(q', \vec{x}') \in Q \times \mathbb{N}^n : (q, \vec{x}) \xrightarrow{*} (q', \vec{x}')\}$ finite?
Variant problems

- **Covering Problem:**
  - **Input:** counter system $S$, $(q, \bar{x})$ and $(q', \bar{x}')$.
  - **Question:** is there a finite run with initial configuration $(q, \bar{x})$ and whose final configuration is $(q', \bar{x}'')$ with $\bar{x}' \preceq \bar{x}''$?

  (control state reachability is an instance with $\bar{x}' = \bar{0}$)

- **Boundedness Problem:**
  - **Input:** counter system $S$ and $(q, \bar{x})$.
  - **Question:** is the set $\{ (q', \bar{x}') \in Q \times \mathbb{N}^n : (q, \bar{x}) \rightarrow^* (q', \bar{x}') \}$ finite?

- **Termination Problem:**
  - **Input:** counter system $S$ and $(q, \bar{x})$.
  - **Question:** is there an infinite run with initial configuration $(q, \bar{x})$?

  Does termination implies boundedness?
What’s next? . . . subclasses

• How to obtain subclasses:

  • restriction on syntactic resources (number of counters, Presburger formulae etc.)

  • restriction on the control graph (e.g. flatness),

  • semantical restrictions (reversal-boundedness, etc.)

• Syntactic presentation of counter systems may be simplified (e.g., avoiding the use of Presburger formulae).
Classes of counter systems

- Reset VASS
  - VASS – L3
    - VAS
  - Succinct CA – L1
    - Minsky Machines
  - Standard CA – L1
    - Reversal-bounded CA – L2
- Affine CS – L2
- Relational CS – L2
  - Flat relational CS
- VASS – L3
- Admissible CS – L2
  - Lossy/Gainy CA – L5
- Reversal-bounded CA – L2
Counter Automata
Standard counter automata

- Standard counter automaton \((Q, n, \delta)\): transitions are of the form either \(q \xrightarrow{\text{inc}(i)} q'\) or \(q \xrightarrow{\text{dec}(i)} q'\) or \(q \xrightarrow{\text{zero}(i)} q'\) where
  - \(\text{inc}(i)\) is a shortcut for \((x'_i = x_i + 1) \land (\bigwedge_{j \neq i} x'_j = x_j)\),
  - \(\text{dec}(i)\) is a shortcut for \((x'_i = x_i - 1) \land (\bigwedge_{j \neq i} x'_j = x_j)\),
  - \(\text{zero}(i)\) is a shortcut for \((x_i = 0) \land (\bigwedge_j x'_j = x_j)\).

- Minsky machines are standard counter automata.
Succinct counter automata

- Each transition either performs zero-tests on a subset of counters or updates counters by adding a vector in $\mathbb{Z}^n$.

- Succinct counter automaton $(Q, n, \delta)$: transitions of the form either $q \xrightarrow{\text{inc}(\vec{b})} q'$ with $\vec{b} \in \mathbb{Z}^n$ or $q \xrightarrow{\text{zero}(\vec{b}')}$ with $\vec{b}' \in \{0, 1\}^n$ where
  - $\text{inc}(\vec{b})$ is a shortcut for $\land_{i \in [1, n]} x'_i = x_i + \vec{b}(i)$,
  - $\text{zero}(\vec{b}')$ is a shortcut for $\land_{i \in [1, n]} \text{s.t. } b'(i) = 1 x_i = 0 \land \land_{i \in [1, n]} x'_i = x_i$

- Morally, standard counter automata and succinct counter automata are identical but there may be differences for complexity issues.
Vector Addition Systems with States (VASS)
What is a VASS?

- **VASS** = finite-state automaton + translations of counters.

- **VASS** is a counter system with transitions of the form
  \[ q \xrightarrow{\vec{b}} q' \text{ with } \vec{b} \in \mathbb{Z}^n, \]  
  which is a shortcut for
  \[ \bigwedge_{i \in [1,n]} x'_i = x_i + \vec{b}(i) \]

- **VAS** = **VASS** with a unique control state.

- Petri nets, **VAS** and **VASS** are equivalent models.
Example
Example

Can $q_0$, \[
\begin{pmatrix}
0 \\
0 \\
20 \\
80
\end{pmatrix}
\] be reached from $q_0$, \[
\begin{pmatrix}
4 \\
20 \\
0 \\
0
\end{pmatrix}
\]?
Decidability/complexity issues

• Theorem: The reachability problem is decidable.  
  [Mayr, STOC 81; Kosaraju, STOC 82]
  
  • No primitive recursive algorithm is known.  
    (use of well quasi-orderings)
  • EXPSPACE-hardness [Lipton, TR 76].

• Theorem: The covering and boundedness problems for VASS are EXPSPACE-complete.  
  [Lipton, TR 76; Rackoff, TCS 78]
  
  • Decidability shown in [Karp & Miller, JCSS 69].
  • EXPSPACE upper bound for path sublogic [Faouzi Atig & Habermehl, RP 09], correcting [Yen, IC 92].

• Checking equality between accessibility sets of two configurations is undecidable [Hack, TCS 76].
A few more remarks

- Control-state reachability is an instance of the covering problem.

- \( \text{EXPSPACE} \)-hardness holds true even with coefficients -1, 0 and 1 only.

- Boundedness and reachability problems are undecidable for VASS with resets.  
  [Dufourd & Finkel & Schnoebelen, ICALP 98].

- Boundedness implies that the transition system from \((q, \vec{x})\) is equivalent to a finite-state automaton.
Conclusion

• Today’s lecture:

  • Classes of counter systems and decision problems.
  
  • Presburger arithmetic.
Conclusion

• Today’s lecture:
  • Classes of counter systems and decision problems.
  • Presburger arithmetic.

• Tomorrow’s lecture: classes with semilinear reachability sets
  • Relational counter systems
  • Reversal-bounded counter automata
  • Flat relational counter automata