Lecture Notes #2: Decidability

Dr. Ferucio Laurenţiu Tiplea
Visiting Professor
School of Computer Science
University of Central Florida
Orlando, FL 32816
E-mail: tiplea@cs.ucf.edu
http://www.cs.ucf.edu/~tiplea
Decidability

1. Introduction to decidability

2. Undecidability

3. Decidability
1. Introduction to decidability

When a formalism is developed, the following questions are crucial:

- **expressive power** – What can I say?
- **decidable questions** – What can I prove?
- **complexity questions** – How hard is to prove it?
- **axiomatics** – How should I prove it?
1. Introduction to decidability

What is an algorithmic problem? An algorithmic problem is a function $f : \mathcal{I} \rightarrow \mathcal{F}$, where $\mathcal{I}$ and $\mathcal{F}$ are two sets at most countable.

As we will only consider algorithmic problems, we will simply call them problems.

$\mathcal{I}$ is called the set of initial data or instances of $f$, and $\mathcal{F}$ is the set of final data.

When $|\mathcal{F}| = 2$ ($\mathcal{F} = \{0, 1\}$ or $\mathcal{F} = \{\top, \bot\}$ or $\mathcal{F} = \{yes, no\}$ etc.), $f$ is called a decision problem; otherwise, it is called a computational problem.
1. Introduction to decidability

Example 1

- $f : \mathbb{N}^2 \to \mathbb{N}$ given by $f(x, y) = x + y$, is a computational problem (the “addition problem”). Each pair $(x, y) \in \mathbb{N}^2$ is an instance of this problem;

- $f : \mathbb{N} \to \{0, 1\}$ given by $f(x) = 1$ if and only if $x$ is a prime, is a decision problem. Each $x \in \mathbb{N}$ is an instance of this problem.
1. Introduction to decidability

Let $f : \mathcal{I} \to \mathcal{F}$ be a problem. As $\mathcal{I}$ and $\mathcal{F}$ are at most countable, they can be encoded as words over a given alphabet $\Sigma$. Therefore, we may assume that $\mathcal{I}, \mathcal{F} \subseteq \Sigma^*$.

**Example 2** Examples of encodings:

- $A \subseteq \mathbb{N} \leadsto \{a^x|x \in A\}$, over $\Sigma = \{a\}$;
- $A \subseteq \mathbb{N}^2 \leadsto \{a^x \# b^y|(x, y) \in A\}$, over $\Sigma = \{a, b, \#\}$;
1. Introduction to decidability

Turing machines are a good model for the study of algorithms, since we can conceive of

• computations with arbitrarily large inputs on their tapes, using an

• arbitrarily large amount of intermediate storage during a computation, and taking an

• arbitrarily large amount of time.

Moreover, Turing machines are universal, in the sense that every known algorithm can be executed by some Turing machine.
1. Introduction to decidability

Consider the following algorithm:

Algorithm $A$

input: $x \in \mathbb{N}$;
output: “yes”, if $x < 5$, and “no”, if $x = 5$;
begin
  $i := x$;
  while $i > 5$ do $i := i + 1$;
  if $i < 5$ then “yes” else if $i = 5$ then “no”;
end.

- $accept(A) = \{0, 1, 2, 3, 4\}$
- $reject(A) = \{5\}$
- $loop(A) = \{x \in \mathbb{N} | x > 5\}$
1. Introduction to decidability

Let $f : \mathcal{I} \rightarrow \{0, 1\}$ be a decision problem, where $\mathcal{I} \subseteq \Sigma^*$. The language associated to $f$ is the set $L_f = \{w \in \mathcal{I} | f(w) = 1\}$.

$f$ is called **decidable** if its language is **recursive**.

\[ f \text{ decidable} \iff \text{there exists an algorithm (Turing machine) that decides } f(L_f) \]

$f$ is called **semi-decidable** if its language is **recursively enumerable**.

\[ f \text{ semi-decidable} \iff \text{there exists an algorithm (Turing machine) that semi-decides } f(L_f) \]

$f$ is called **undecidable** if it is not decidable.
1. Introduction to decidability

A decision problem \( f : \mathcal{I} \to \{0, 1\} \) is reducible to a decision problem \( g : \mathcal{I}' \to \{0, 1\} \), abbreviated \( f \prec g \), if there exists an algorithm (Turing machine) \( M \) such that:

- \((\forall x \in \mathcal{I})(M(x) \in \mathcal{I}')\);
- \((\forall x \in \mathcal{I})(f(x) = 1 \iff g(M(x)) = 1)\).

**Proposition 1** Let \( f \) and \( g \) be decision problems.

- If \( f \prec g \) and \( g \) is decidable, then \( f \) is decidable.
- If \( f \prec g \) and \( f \) is undecidable, then \( g \) is undecidable.
2. Undecidability

2.1. The halting problem
2.2. Rice’s theorem revised
2.3. Post correspondence problem
2.4. Domino problems
2.5. Hilbert’s 10th problem and consequences
2.6. The word problem for finitely presented monoids
2.7. Valid and invalid computations
2.8. Greibach’s theorem and applications
2.1. The Halting Problem

2.1.1. The halting problem and its undecidability

2.1.2. Stack machines. Counter machines

2.1.3. Applications to Petri nets

2.1.4. Applications to security protocols
2.1.1. The Halting Problem and its Undecidability

The halting problem for a given algorithmic formalism is the problem of whether or not a given procedure of the formalism when executed with a given input eventually terminates.

The Halting Problem

**Instance:** algorithm $\mathcal{A}$ (Turing machine $M$) and input $x$;

**Question:** does $\mathcal{A}(M)$ halt on $x$?

**Theorem 1** The halting problem for Turing machines is undecidable.
2.1.1. The Halting Problem and its Undecidability

**Proof**  Assume that there exists an algorithm $A$ that decides the halting problem. Denote by $\langle B \rangle$ an arbitrary but fixed encoding of an algorithm $B$. Let $D$ be the following algorithm:

```
Algorithm $D$
input: algorithm $B$;
output: 0 if $B(\langle B \rangle) \uparrow$;
begin
  $y := A(B, \langle B \rangle)$;
  if $y = 0$ then 0 else loop forever
end.
```

It is easy to see that $D(\langle D \rangle) \uparrow \iff D(\langle D \rangle) \downarrow$, which is a contradiction. 

$\square$
2.1.1. The Halting Problem and its Undecidability

There are several variants on the halting problem.

**The Empty-input Halting Problem**

Instance: algorithm $\mathcal{A}$ (Turing machine $M$);

Question: does $\mathcal{A} (M)$ halt on the empty-input?

**Corollary 1** The empty-input halting problem for Turing machines is undecidable.

**Proof** We exhibit a reduction from the halting problem:

$$(\mathcal{A}, x) \rightsquigarrow \mathcal{A}'$$

where $\mathcal{A}'$, on the empty-input, generates $x$ and then simulates $\mathcal{A}$ on $x$. 
2.1.1. The Halting Problem and its Undecidability

Given an algorithm $\mathcal{A}$ and an input $x$ for it, define the algorithm $\mathcal{A}'$ as follows:

**Algorithm $\mathcal{A}'$**

input: none;
output: $z = \mathcal{A}(x)$;
begin
  $z := \mathcal{A}(x)$;
end.

Clearly, $\mathcal{A}(x) \downarrow$ iff $\mathcal{A}' \downarrow$. \hfill $\Box$
2.1.1. The Halting Problem and its Undecidability

The Uniform Halting Problem

Instance: algorithm $A$ (Turing machine $M$);
Question: does $A(M)$ halt on all inputs?

Corollary 2 The uniform halting problem for Turing machines is undecidable.

Proof We exhibit a reduction from the halting problem:

$$(A, x) \leadsto A'$$

where $A'$, on an arbitrary input $y$, erases $y$, generates $x$, and then simulates $A$ on $x$. 
2.1.1. The Halting Problem and its Undecidability

Given an algorithm $A$ and an input $x$ for it, define the algorithm $A'$ as follows:

**Algorithm $A'$**

- **input:** $y$
- **output:** $z = A(x)$
- **begin**
  - $z := A(x)$;
- **end.**

Clearly, $A(x) \downarrow$ iff $(\forall y)(A'(y) \downarrow)$.  \hfill $\Box$
2.1.2. Multistack Machines. Counter Machines

A \textit{k-stack machine}, abbreviated k-SM, is a 7-tuple \((Q, \Sigma, \Gamma, \delta, q_0, Z, F)\), where:

- \(Q\) is a non-empty finite set of \textit{states}
- \(q_0 \in Q\) is the \textit{initial state}
- \(F \subseteq Q\) is the \textit{final set of states}
- \(\Sigma\) is the \textit{input alphabet}
- \(\Gamma\) is the \textit{stack alphabet}
- \(Z \in \Gamma \setminus \Sigma\) is the \textit{bottom-of-stack marker}
- \(\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma)^k \rightarrow Q \times (\Gamma^*)^k\) is the \textit{transition function}

satisfying the property that the bottom-of-stack marker \(Z\) “cannot be erased” and it “cannot appear elsewhere on the stacks”.
2.1.2. Multistack Machines. Counter Machines

- a b a d c $
- input tape - read-only
- $ - endmarker
- Z - bottom-of-stack marker

Stack machine
2.1.2. Multistack Machines. Counter Machines

Computation relation:

\[(q, u|av, Zu_1X_1, \ldots, Zu_kX_k) \vdash (q', ua|v, \gamma_1, \ldots, \gamma_k)\]

iff

\[\delta(q, a, X_1, \ldots, X_k) = (q', \gamma_1, \ldots, \gamma_k)\]

where \(u, v \in \Sigma^*, a \in \Sigma \cup \{\lambda\}, u_1X_1, \ldots, u_kX_k \in \Gamma^*\).

**Theorem 2** A language is accepted by a Turing machine iff it is accepted by a 2-stack machine.

**Corollary 3** The halting problem for 2-stack machines is undecidable.
A **k-counter machine**, abbreviated k-CM, is a k-SM $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ such that $|\Gamma| = 2$.

**Theorem 3** A language is accepted by a Turing machine iff it is accepted by a 2-CM.

**Corollary 4** The halting problem for 2-counter machines is undecidable.
2.1.2. Multistack Machines. Counter Machines

A ”simplified’ version of counter machines:

\[ M = (Q, q_0, q_f, C, x_0, I), \]

where:

- \( Q \) is a non-empty finite set of states;
- \( q_0 \in Q \) is the initial state, and \( q_f \in Q \) is the final state;
- \( C \) is a finite set of counters, each of which being able to hold a natural number;
- \( x_0 : C \rightarrow \mathbb{N} \) is the initial content of counters;
- \( I \) is a finite set of instructions. For each state there is at most an instruction that can be executed at that state; for \( q_f \) there is no instruction. Each instruction is one of the following forms:
2.1.2. Multistack Machines. Counter Machines

- **increment instruction** \( I(q, c, q') \)

  \( q : \) begin
  \hspace{1cm} \( c := c + 1; \)
  \hspace{1cm} go to \( q' \)
  end

- **test instruction** \( I(q, c, q', q'') \)

  \( q : \) begin
  \hspace{1cm} if \( c = 0 \) then go to \( q' \)
  \hspace{1cm} else begin
  \hspace{2cm} \( c := c - 1; \)
  \hspace{2cm} go to \( q'' \)
  \hspace{1cm} end
  end
A configuration is a pair \((q, x)\), where \(q \in Q\) and \(x : C \rightarrow N\). A configuration \((q, x)\) is called initial if \(q = q_0\) and \(x = x_0\). A configuration \((q, x)\) is called final if \(q = q_f\).

Computation:

\[(q, x) \vdash (q', x')\]

iff one of the following holds:

- there exists \(I(q, c, q')\) such that \(x'(c) = x(c) + 1\) and \(x'(c') = x(c')\), for all \(c' \in C - \{c\}\);

- there exists \(I(q, c, q_1, q_2)\) such that
  - if \(x(c) = 0\), then \(q' = q_1\) and \(x' = x\);
  - if \(x(c) \neq 0\), then \(q' = q_2\), \(x'(c) = x(c) - 1\), and \(x'(c') = x(c)\), for all \(c' \in C - \{c\}\).
2.1.3. Applications to Petri Nets

Petri nets, abbreviated PN, have been introduced by Carl Adam Petri in 1962 as models of distributed systems, where concurrency and communication play an important role.

A PN is a system $\Sigma = (S, T, F, W)$, where:

- $S$ is a finite non-empty set of **places**;
- $T$ is a finite non-empty set of **transitions**;
- $S \cap T = \emptyset$;
- $F \subseteq S \times T \cup T \times S$ is the **flow relation**;
- $W : S \times T \cup T \times S \rightarrow \mathbb{N}$ is the **weight function** satisfying $W(x, y) = 0$ iff $(x, y) \notin F$. 

UCF-SCS/COT5310/Fall 2005/F.L. Tiplea 25
Configurations in Petri net theory are called *markings*, and they are defined as functions \( M : S \to \mathbb{N} \).

Because \( S \) is a finite set, markings are usually represented as \( S \)-dimensional vectors.

**Computation (firing) rule:**

- A transition \( t \) is **enabled at** \( M \), denoted \( M[t] \), if
  \[
  W(s, t) \geq M(s),
  \]
  for all \( s \in S \);

- If \( t \) is enabled at \( M \) then \( t \) may **fire** yielding a new marking \( M' \) given by
  \[
  M'(s) = M(s) - W(s, t) + W(t, s),
  \]
  for all \( s \in S \). We denote this by \( M[t]M' \).
2.1.3. Applications to Petri Nets

Graphical representation:

**Example 3** Vending machine:

\[ \(1, 0, 0, 0, 2, 1\)[\(t_1\)](0, 1, 1, 0, 2, 1)[\(t_3\)](0, 1, 0, 1, 2, 1)[\(t_4\)](1, 1, 0, 0, 3, 0) \]

- \(s_1 = \text{ready}\)
- \(s_2 = \text{counter}\)
- \(s_3 = \text{inserted}\)
- \(s_4 = \text{accepted}\)
- \(s_5 = \text{warm}\)
- \(s_6 = \text{cold}\)
- \(t_1 = \text{insert}\)
- \(t_2 = \text{reject}\)
- \(t_3 = \text{accept}\)
- \(t_4 = \text{dispense}\)
- \(t_5 = \text{brew}\)
2.1.3. Applications to Petri Nets

A pair $\gamma = (\Sigma, M_0)$, where $\Sigma$ is a Petri net and $M_0$ is a marking of $\Sigma$ is called a marked Petri net.

A marking $M$ is reachable in $\gamma$ if there exists a sequence of transitions $w \in T^*$ such that $M_0[w]\triangleright M$.

A marking $M$ is coverable in $\gamma$ if there exists a reachable marking $M'$ in $\gamma$ such that $M' \geq M$ (the inequality on vectors is componentwise understood).

$\gamma$ is bounded if there exists $n \in \mathbb{N}$ such that $M(s) \leq n$, for any $s \in S$ and reachable marking $M$.

A transition $t$ of $\gamma$ is live if for any reachable marking $M$ there exists $M'$ reachable from $M$ such that $M'[t]$. If all transitions are live, the $\gamma$ is called live.
2.1.3. Applications to Petri Nets

Basic decision problems in Petri net theory: reachability, coverability, boundedness, and liveness.

All these problems are decidable for Petri nets (details will be provided in a separate section). However, they are undecidable for almost all Petri net extensions. For instance, we will prove that they are undecidable for inhibitor Petri nets (see InhibitorPetriNets.pdf).
2.1.4. Applications to Security Protocols

see SecurityProtocols.pdf
2.3. Post’s Correspondence Problem

2.3.1. Post’s correspondence problem
2.3.2. Applications to first-order logic
2.3.3. Applications to formal language theory
2.3.1. Post’s Correspondence Problem


**Post’s Correspondence Problem (PCP)**

**Instance:** list of pairs of words \( L = \{(u_1, v_1), \ldots, (u_n, v_n)\} \)

**Question:** Is there any list of numbers \( i_1, \ldots, i_k \) s.t.

\[
u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}\]

Any list of numbers \( i_1, \ldots, i_k \) such that

\[
u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}\]

is called a solution of \( L \).
2.3.1. Post’s Correspondence Problem

**Example 4** The list 1,2,1,3 is a solution to the PCP instance

\[ L = \{(a^2, a^2b), (b^2, ba), (ab^2, b)\} \]

**Example 5** The PCP instance

\[ L = \{(a^2b, a^2), (a, ba^2)\} \]

has no solution.

**Example 6** The list 1,3,2,3 is a solution to the PCP instance

\[ L = \{(1, 101), (10, 00), (011, 11)\} \]
2.3.1. Post’s Correspondence Problem

**Proposition 2** The PCP instance

\[ L = \{(a^{k_1}, a^{l_1}), \ldots, (a^{k_n}, a^{l_n})\} \]

has solutions if and only if

1. there exists \( i \) such that \( k_i = l_i \), or
2. there exist \( i \) and \( j \) such that \( k_i > l_i \) and \( k_j < l_j \).

**Corollary 5** PCP over one-letter alphabets is decidable.
2.3.1. Post’s Correspondence Problem

**Proposition 3** Any PCP instance over an alphabet \( \Sigma \) with \( |\Sigma| \geq 2 \) is equivalent to a PCP instance over an alphabet \( \Delta \) with \( |\Delta| = 2 \).

**Proof** Assume \( \Sigma = \{a_1, \ldots, a_n\} \) and \( n > 2 \). Let \( \Delta = \{a, b\} \), where \( a \neq b \).

Encode \( a_i \) by \( ba^ib \), for any \( i \).

Define:

- \( PCP_1 \) – PCP instances over one-letter alphabets
- \( PCP_2 \) – PCP instances over two-letter alphabets
2.3.1. Post’s Correspondence Problem

**Theorem 4**  PCP is undecidable.

**Proof**  Show that the halting problem for Post machines, which are equivalent to Turing machines, can be reduced to PCP.

**Corollary 6**  $PCP_2$ is undecidable.

**Summary:**
- $PCP_1$ is decidable
- $PCP_2$ is undecidable
2.3.1. Post’s Correspondence Problem

Other variations:

- $PCP(n)$ – PCP instances of length $n$
  \[
  (L = \{(u_1, v_1), \ldots, (u_n, v_n)\})
  \]

- $PCP_1(n)$
- $PCP_2(n)$

$PCP_1(n)$ is decidable, for all $n$. 
2.3.1. Post’s Correspondence Problem

**Theorem 5** (Ehrenfeucht, Karhumaki, Rozenberg, 1982)
PCP(2) is decidable.

For a simpler proof than the original one see [HaHH2000.pdf](#).

**Theorem 6** Matiyasevich, Senizergues, 1996)
PCP(7) is undecidable.

**Proof** See [MaSe1996.pdf](#).

**Open problems:** PCP(3),...,PCP(6)
### 2.3.2. Applications of PCP to First-order Logic

**Validity problem for first-order logic (VPFOL)**
- **Instance:** First-order formula $\phi$
- **Question:** Is $\phi$ valid?

**Satisfiability problem for first-order logic (SPFOL)**
- **Instance:** First-order formula $\phi$
- **Question:** Is $\phi$ satisfiable?

These two decision problems are equivalent because

$$\phi \text{ is valid } \iff \neg \phi \text{ is not satisfiable}$$
Theorem 7 VPFOL is undecidable.

Proof Reduce PCP$_2$ to VPFOL. Given a PCP$_2$ instance

\[ L = \{(u_1, v_1), \ldots, (u_n, v_n)\} \]

over $\Sigma = \{0, 1\}$, define a formula $\phi$ such that

$L$ has solutions $\iff$ $\phi$ is valid

$\phi$ is defined as follows:

- let $a$ be a constant. It will interpreted by $\lambda$ in some interpretation $\mathcal{I}$;
- let $f_0$ and $f_1$ be function symbols. They will be interpreted by $\mathcal{I}(f_0)(x) = x_0$ and $\mathcal{I}(f_1)(x) = x_1$. We will simply write $f_{b_1 \ldots b_k}(x)$ instead of $f_{b_k}(\cdots f_{b_1}(x) \cdots)$;
2.3.2. Applications of PCP to First-order Logic

- let $P$ be a predicate symbol. It will be interpreted by
  \[ I(P)(x, y) \iff x, y \in \Sigma^* \land x = u_{i_1} \cdots u_{i_k} \land y = v_{i_1} \cdots v_{i_k}, \]
  for some $i_1, \ldots, i_k$

- let $\phi_1$ be the formula \( \phi_1 = \land_{i=1}^{k} P(f_{u_i}(a), f_{v_i}(a)) \)
- let $\phi_2$ be the formula \( \phi_2 = (\forall u, v)(P(u, v) \Rightarrow \land_{i=1}^{k} P(f_{u_i}(u), f_{v_i}(u)) \)
- let $\phi_3$ be the formula \( \phi_3 = (\exists x)(P(x, x)) \)
- let $\phi$ be the formula \( \phi = (\phi_1 \land \phi_2 \Rightarrow \phi_3) \)

Then,

\[ L \text{ has solutions} \iff \phi \text{ is valid} \]

which concludes the proof. \( \square \)
2.3.3 Applications of PCP to Formal Language Theory

Intersection problem for CFL (IPCFL)

Instance: context-free grammars $G_1$ and $G_2$
Question: Is $L(G_1) \cap L(G_2) \neq \emptyset$?

**Theorem 8** IPCFL is undecidable.

**Proof** Reduce PCP to IPCFL. Given a PCP instance

$$L = \{ (u_1, v_1), \ldots, (u_n, v_n) \}$$

over $\Sigma$, define two CF-grammars $G_1$ and $G_2$ such that

$L$ has solutions $\iff L(G_1) \cap L(G_2) \neq \emptyset$

The grammars are:

- $G_1 : S \rightarrow iSu_i | iu_i$, for all $i$
- $G_2 : S \rightarrow iSv_i | iv_i$, for all $i$

$\square$
2.3.3. Applications of PCP to Formal Language Theory

Equivalence problem for CFG (EPCFG)

Instance: context-free grammars $G_1$ and $G_2$

Question: Is $L(G_1) = L(G_2)$?

**Theorem 9** EPCFG is undecidable.

**Proof** Reduce $\neg$PCP to EPCFG. Given a PCP instance

$$L = \{(u_1, v_1), \ldots, (u_n, v_n)\}$$

over $\Sigma$, define two CF-grammars $G_1$ and $G_2$ such that

$L$ has no solution $\iff L(G_1) = L(G_2)$

Define two grammars $G_1$ and $G_2$ such that

- $G_1$ generates $L(G_1) = \{1, \ldots, n\}^* \# \Sigma^*$, where $\#$ is a new symbol;
2.3.3. Applications of PCP to Formal Language Theory

- $G_2$ generates $L(G_2) = (L(G_1) - A) \cup (L(G_1) - B)$, where
  - $A = \{i_1 \cdots i_k \# u_{i_k} \cdots u_{i_1} | i_1, \ldots, i_k \in \{1, \ldots, n\}\}$
  - $B = \{i_1 \cdots i_k \# v_{i_k} \cdots v_{i_1} | i_1, \ldots, i_k \in \{1, \ldots, n\}\}$

It is easy to see that two context-free grammars $G_1$ and $G_2$ as above exist, and $L$ has no solution iff $L(G_1) = L(G_2)$. □
Applications of PCP to Formal Language Theory

Ambiguity problem for CFG (APCFG)
 Instance: context-free grammar $G$
 Question: Is $G$ ambiguous?

**Theorem 10** APCFG is undecidable.

**Proof** Reduce PCP to APCFG. Given a PCP instance

$L = \{(u_1, v_1), \ldots, (u_n, v_n)\}$

over $\Sigma$, define a CFG $G$ such that

$L$ has solutions $\Leftrightarrow$ $G$ is ambiguous

Define $G$ by

- $S \rightarrow S_1 | S_2$
- $S_1 \rightarrow u_i S_1 i | u_i i$
- $S_2 \rightarrow v_i S_2 i | v_i i$

for all $i$. □
A semi-group \((S, \cdot)\) is called **finitely presented** if there exists a finite set \(A\) of generators for \(S\) and a finite set \(E\) of equations over \(A\) (i.e., pairs of words over \(A\)) such that any valid equation in \(S\) can be obtained by derivation from \(E\). That is, if \(t = t'\) is valid in \(S\), then \(t \Rightarrow_E t'\).

**Example 7** Let \(S\) be a semi-group generated by \(A = \{a_1, a_2, a_3\}\) under the equations

- \(a_2a_1 = a_1a_2\)
- \(a_3a_2 = a_2a_2a_3\)
- \(a_3a_1 = a_1\).

Then, \(a_1a_2a_2 = a_1a_2\) is valid in \(S\).
2.6. The word problem for finitely presented monoids

Word Problem for Semi-groups (WPSG)

Instance: finite semi-group presentation \((A, E)\) and equation \(t = t'\)

Question: Does \(t = t'\) hold true in the semi-group presented by \((A, E)\)?

This problem was shown to be undecidable in:


The problem can be reduced to the reachability problem for Thue systems.
2.6. The word problem for finitely presented monoids


A *Thue system* over an alphabet $\Sigma$ is any set of unordered pairs of words over $\Sigma$. Each pair $\{t, t'\}$ is usually written as $t = t'$.

A *semi-Thue system* over an alphabet $\Sigma$ is any set of ordered pairs of words over $\Sigma$. Each pair $(t, t')$ is usually written as $t \rightarrow t'$.

**Reachability Problem for Semi-Thue Systems (RPSTS)**

**Instance:** semi-Thue system $R$ and words $t$ and $t'$

**Question:** Does $t \Rightarrow^* R t'$?
Theorem 11 The reachability problem for (semi-)Thue systems is undecidable.

Proof Reduce the halting problem for Turing machines to this problem. □

Corollary 7 The word problem for finitely presented semi-groups (monoids) is undecidable.
2.6. The word problem for finitely presented monoids

Term rewriting systems and related problems:

TermRewritingSystems.pdf
3. Decidability

Techniques for proving decidability:

- **reducibility**: if a problem $A$ is reducible to a problem $B$ and $B$ is decidable, then $A$ is decidable;
- **ad hoc techniques**.
3. Decidability

Coverability tree based techniques

General remarks:

- a coverability tree reduces the analysis of an infinite state space to the analysis of a finite state space;
- cut off infinite branches and add extra information to the leaf nodes;
- some properties of the original state space (reachability tree) may be lost.

We illustrate the technique on vector addition systems.
3. Decidability

A vector addition system (VAS) is a couple $\mathcal{W} = (W, v_0)$, where:

- $W$ is a finite set of $n$-dimensional vectors with integer components ($W = \{v_1, \ldots, v_k\} \subseteq \mathbb{Z}^n$);
- $v_0$ is an $n$-dimensional vector with positive integer components ($v_0 \in \mathbb{N}^n$).

Example 8 $\mathcal{W} = (W, v_0)$, where

$$W = \{(-1, 1, 0, 1), (0, -1, 0, 0), (1, 0, 0, -1), (0, 0, -1, 1)\}$$

and $v_0 = (1, 0, 0, 1)$, is a vector addition system.
3. Decidability

Let $W = (W, v_0)$ be a VAS, $x \in \mathbb{Z}^n$, and $v \in W$.

- $v$ is **enabled at** $x$, denoted $x[v]$ or $x \rightarrow v$, if $x + v \geq 0$. $W(x)$ stands for the set $\{v \in W | x[v]\}$;
- if $v$ is enabled at $x$ then $v$ may be applied yielding a new vector $x'$ given by $x' = x + v$. We denote this by $x[v]x'$ or $x \rightarrow v x'$;
- $\Rightarrow = \bigcup_{v \in W} v$;
- $\Rightarrow$ is the reflexive and transitive closure of $\Rightarrow$;
- $x$ is **reachable** in $W$ if $d \Rightarrow x$;
- $[v_0]$ is the set of all reachable vectors in $W$, called the **reachability set** of $W$;
- $x$ is **coverable** in $W$ if $v_0 \Rightarrow x'$ and $x' \geq x$, for some $x'$;
- $v$ is **dead** in $W$ if $\neg (x[v])$, for any $x$ reachable in $W$. 
3. Decidability

Let $\mathcal{W} = (W, v_0)$ be a VAS. A labeled tree $\mathcal{R} = (V, E, l_1, l_2)$ is a **reachability tree** of $\mathcal{W}$ if:

1. its root $x_0$ is labeled by $v_0$, i.e., $l_1(x_0) = v_0$;
2. $\forall x \in V, |x^+| = |W(l_1(x))|$;
3. $\forall x \in V$ with $|x^+| > 0$ and $\forall v \in W(l_1(x))$ there exists $x' \in x^+$ such that:
   - (a) $l_1(x') = l_1(x) + v$;
   - (b) $l_2(x, x') = v$.

Any two reachability trees of $\mathcal{W}$ are isomorphic. Therefore, we may talk about the **reachability tree** of $\mathcal{W}$, denoted $\mathcal{R}(\mathcal{W})$. 
3. Decidability

**Proposition 4** Let $\mathcal{W} = (W, v_0)$ be a VAS. Then,

1. $\mathcal{R}(\mathcal{W})$ is finitely branched;
2. $x$ in $\mathcal{R}(\mathcal{W})$ is a leaf node iff no vector in $W$ is enabled at $l_1(x)$;
3. $[v_0] = \{l_1(x) | x \in V\}$.

$\mathcal{R}(\mathcal{W})$ may be infinite even if $[v_0]$ is finite!
3. Decidability

We will derive a finite structure from $\mathcal{R}(\mathcal{W})$.

Let $\omega \not\in \mathbb{Z}$ and $\mathbb{Z}_\omega = \mathbb{Z} \cup \{\omega\}$. Extend $+$ and $<$ to $\mathbb{Z}_\omega$ by:

- $n + \omega = \omega + n = \omega$, for any $n \in \mathbb{Z}$;
- $n < \omega$, for any $n \in \mathbb{Z}$.

The notation $x[v]$ etc. is usually extended to vectors over $\mathbb{Z}_\omega$. 

3. Decidability

Let $\mathcal{W} = (W, v_0)$ be a VAS. A labelled tree $T = (V, E, l_1, l_2)$ is a coverability tree of $\mathcal{W}$ if:

1. its root $x_0$ is labelled by $v_0$, i.e., $l_1(x_0) = v_0$;
2. $\forall x \in V$,
   \[
   |x^+| = \begin{cases} 
   0, & W(l_1(x)) = \emptyset \text{ or } \\
   |W(l_1(x))|, & \text{otherwise}
   \end{cases}
   \]
   \[
   (\exists x' \in d_T(x_0, x))(x \neq x' \land l_1(x) = l_1(x'))
   \]
3. $\forall x \in V$ with $|x^+| > 0$ and $\forall v \in W(l_1(x))$ there exists $x' \in x^+$ such that:
   (a) $l_1(x')(i) = \omega$ if $(\exists x'' \in d_T(x_0, x))(l_1(x'') \leq l_1(x) + v \land l_1(x'')(i) < (l_1(x) + v)(i))$, and $l_1(x')(i) = (l_1(x) + v)(i)$, otherwise (for any $i$);
   (b) $l_2(x, x') = v$. 
Any two coverability trees of $\mathcal{W}$ are isomorphic. Therefore, we may talk about the coverability tree of $\mathcal{W}$, denoted $\mathcal{T}(\mathcal{W})$.

**Proposition 5** Let $\mathcal{W} = (W, v_0)$ be a VAS and $\mathcal{T}(\mathcal{W}) = (V, E, l_1, l_2)$ its coverability tree. Then:

1. $\mathcal{T}(\mathcal{W})$ is finitely branched;
2. $x$ in $\mathcal{T}(\mathcal{W})$ is a leaf node iff $W(l_1(x)) = \emptyset$ or there exists $x' \in d_\mathcal{T}(x_0, x)$ such that $x \neq x'$ and $l_1(x) = l_1(x')$;
3. let $x_{i_0}, x_{i_1}, \ldots, x_{i_m}$ be pairwise distinct nodes such that $x_{i_j} \in d_\mathcal{T}(\gamma)(x_0, x_{i_{j+1}})$, for any $0 \leq j \leq m - 1$.
   (a) if $l_1(x_{i_0}) = l_1(x_{i_1}) = \cdots = l_1(x_{i_m})$, then $m \leq 1$;
   (b) if $l_1(x_{i_0}) < l_1(x_{i_1}) < \cdots < l_1(x_{i_m})$, then $m \leq n$;
4. $\mathcal{T}(\gamma)$ is finite.
3. Decidability

**Theorem 12** Let $\mathcal{W} = (W, v_0)$ be a VAS and $\mathcal{T}(\mathcal{W}) = (V, E, l_1, l_2)$ its coverability tree. Then, a vector $x$ is coverable in $\mathcal{W}$ iff it is coverable in $\mathcal{T}(\mathcal{W})$.

**Corollary 8** Coverability, deadness and finiteness problems are decidable for VASs.