3. Coverability is EXPSPACE-hard, Lipton '76

Presentation follows the survey article

Decidability and Complexity of Petri Net Problems – An Introduction

by Javier Esparza '96.

Rule of thumb:

- All interesting questions about the behavior of (general) Petri nets are EXPSPACE-hard.
- More precisely, they require at least $2^{O(n)}$ space where n is the size of the input.

Follows (for all problems) from one fact:

Reduction:

- A deterministic exponentially-bounded Turing machine of size n can be simulated by a PN of size $O(n)$
- There is a polynomial-time procedure to construct this net.

Problem / Solution:

- Turing machines and Petri nets do not fit well.
- Known:

Bounded Turing machines $\overset{(1)}{\leq}$ (can be simulated by) Bounded counter programs $\overset{(2)}{\leq}$ Petri nets.

(Lipton's result)

Concerning (1):

There is a polynomial-time procedure that

accepts a deterministic Turing machine $\mathcal{A}$ of size n
and returns a counter program $C_{\mathcal{A}}$ with $O(n)$ commands
that satisfies the following

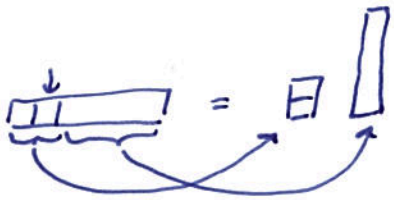
↳ $C_{\mathcal{A}}$ simulates the computation of $\mathcal{A}$ on the empty tape.

↳ In particular: $C_{\mathcal{A}}$ halts iff $\mathcal{A}$ halts.

↳ Moreover: if $\mathcal{A}$ is exponentially-bounded,
then $C_{\mathcal{A}}$ is 2^{2^n}-bounded.

Construction:

↳ A Turing machine over $\{0,1\}$ can be simulated
by two stacks (over $\{0,1\}$):



- cut the tape in half
- moving the head left
is mimicked by
popping a bit from the left stack
and pushing it onto the right stack.

↳ A stack over $\{0,1\}$ can be simulated by two counters

- One counter holds the value
represented in binary by the bits on the stack
(least significant bit on top).

- Pushing a 1 onto a stack representing value $v \in \mathbb{N}$
is implemented by computing

$$2v+1.$$

↳ A stack over $\{0,1\}$ of size at most 2^{n^k}
can be simulated by two counters
that are bounded by $2^{2^{n^{k'}}}$.

Hence:

- A 1-tape Turing machine with space 2^{n^k}
can be simulated by 4 counters bounded by $2^{2^{n^{k'}}}$.
- Minsky '67: ↳ A Turing machine can be simulated by a 2-counter machine
↳ But the values need one more exponential.

Concerning (2): Goal of this lecture

↳ Using (1), it is sufficient
to simulate a 2^{2^n}-bounded counter program of size $O(n)$
by a PN of size $O(n)$.

↳ We develop such an encoding.

3.1 Petri Net Programs

↳ Encode counter programs into PN programs

↳ A PN program is a convenient description of a PN

↳ The PN semantics (translation from PN programs to PN)
is easy:
every command yields a transition.

Commands:

l: $x := x+1$	
l: $x := x-1$	(only if $x>0$)
l: goto l_1	// unconditional jump
l: goto l_1 or goto l_2	// non-deterministic jump
l: gosub l_1	// subroutine call
l: return	// subroutine return
l: halt	// stop execution.

Syntactic correctness:

↳ Decompose a program into
- main program that only calls 1st level subroutines
- 1st level subroutines only call 2nd level subroutines

etc.

↳ All programs given here are well-structured (in this sense).

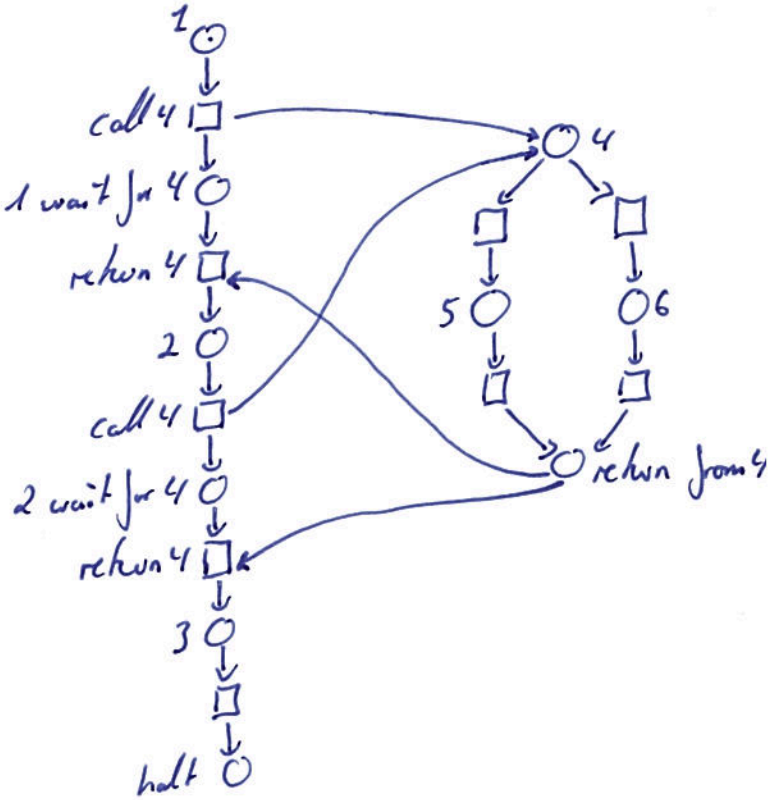
PN Semantics:

↳ Immediate for all commands except subroutine calls/returns

↳ Semantics of subroutines on an example

main
1: gosub 4;
2: gosub 4;
3: halt;

1st level subroutine
4: goto 5 or goto 6;
5: return;
6: return;



Note:

↳ The same part of the PN (subroutine 4) can be used several times.

↳ This saves exponential space / keeps the PN exponentially smaller.

↳ Illustration:

routine a calls 2x routine b
routine b calls 2x routine c
routine c calls 2x routine d
} a calls 8x d.

Size of the PN Semantics

- PN Semantics for a PN program with k commands has $O(k)$ places, $O(k)$ transitions, and $O(k)$ edges.
- The initial marking is also of size $O(k)$.
  ↳ Altogether, the PN for the PN program has a size of $O(k)$.

3.2 Construction

Goal:
- Consider a 2^{2^n}-bounded counter program C with $O(n)$ commands.
- Construct a PN program NP_C with $O(n)$ commands that simulates C.
- PN program NP_C corresponds to a PN of size $O(n)$.

The notion of simulation:
- Note that C is deterministic while NP_C will be non-deterministic.
- Ensure
  C halts $\Leftrightarrow$ some computation of NP_C halts (to be distinguished from fail).

Technically:
- Each variable x of NP_C has a complement variable $\bar{x}$.
- The following invariant will hold:
  $$\bar{x} = 2^{2^n} - x.$$
- The program is as follows
  $$NP_C = NP_C^{init}; NP_C^{sim}$$
  - NP_C^{init} sets $\bar{x}$ to 2^{2^n}
  - NP_C^{sim} simulates C, preserving the invariant.