Pushdown systems
A small program (where $n \geq 1$):

```c
bool g=true;
void main() {
    level_1();
    level_1();
}
void level_i() {
    level_{i+1}();
    level_{i+1}();
}
void level_n() {
    g:=not g;
}
```

Question: Is $g$ true when the program terminates?
Example 1 has got finitely many states.
(The call stack is bounded by $n$.)

Can be treated by “inlining” (replace procedure calls by a copy of the callee).

Inlining causes an exponential state-space explosion.

Inlining is inefficient: every copy of each procedure will be investigated separately.

Inlining not applicable for recursive procedure calls.
Example 2: Drawing skylines

procedure $p$;
$p_0$: if $?$ then
$p_1$: call $s$; right;
$p_2$: if $?$ then call $p$; end if;
else
$p_3$: up; call $p$; down;
end if
$p_4$: return

procedure $s$;
$s_0$: if $?$ then return; end if;
$s_1$: up; call $p$; down;
$s_2$: return

procedure $main$;
$m_0$: call $s$;
$m_1$: return;

$S = \{p_0, \ldots, p_4, s_0, \ldots, s_2, m_0, m_1\}^*$, initial state $m_0$
Example 2 has got infinitely many states.

Inlining not applicable!

Cannot be analyzed by naїvely searching all reachable states.

We shall require a finite representation of infinitely many states.
void quicksort (int left, int right) {
    int lo, hi, piv;
    if (left >= right) return;
    piv = a[right]; lo = left; hi = right;
    while (lo <= hi) {
        if (a[hi] > piv) {
            hi = hi - 1;
        } else {
            swap a[lo], a[hi];
            lo = lo + 1;
        }
    }
    quicksort(left, hi);
    quicksort(lo, right);
}
**Question:** Does Example 3 sort correctly? Is termination guaranteed?

The mere structure of Example 3 does not tell us whether there are infinitely many reachable states:

- *finitely* many if the program terminates
  
- *infinitely* many if it fails to terminate

Termination can only be checked by directly dealing with infinite state sets.
A computation model for procedural programs

Control flow:

- sequential program (no multithreading)
- procedures
- mutual procedure calls (possibly recursive)

Data:

- global variables (restriction: only finite memory)
- local variables in each procedure (one copy per call)
Pushdown systems

A pushdown system (PDS) is a triple \((P, \Gamma, \Delta)\), where

- \(P\) is a finite set of control states;
- \(\Gamma\) is a finite stack alphabet;
- \(\Delta\) is a finite set of rules.
Rules have the form $pA \rightarrow qw$, where $p, q \in P$, $A \in \Gamma$, $w \in \Gamma^*$. 

Like acceptors for context-free language, but without any input!
Behaviour of a PDS

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS and $c_0 \in P \times \Gamma^*$.  

With $\mathcal{P}$ we associate a transition system $\mathcal{T}_\mathcal{P} = (S, \rightarrow, r)$ as follows:

$S = P \times \Gamma^*$ are the states (which we call configurations);

we have $pA w' \rightarrow q w w'$ for all $w' \in \Gamma^*$ iff $pA \leftarrow q w \in \Delta$;

$r = c_0$ is the initial configuration.
Transition system of a PDS

\[ pA \leftrightarrow qB \]
\[ pA \leftrightarrow pC \]
\[ qB \leftrightarrow pD \]
\[ pC \leftrightarrow pAD \]
\[ pD \leftrightarrow p\varepsilon \]
Procedural programs and PDSs

$P$ may represent the valuations of global variables.

$\Gamma$ may contain tuples of the form (program counter, local valuations)

Interpretation of a configuration $pA w$:

- global values in $p$, current procedure with local variables in $A$
- “suspended” procedures in $w$

Rules:

- $pA \rightarrow qB \equiv$ statement within a procedure
- $pA \rightarrow qBC \equiv$ procedure call
- $pA \rightarrow q\varepsilon \equiv$ return from a procedure
Let $\mathcal{P}$ be a PDS and $c, c'$ two of its configurations.

**Problem:** Does $c \rightarrow^* c'$ hold in $\mathcal{T}_\mathcal{P}$?

**Note:** $\mathcal{T}_\mathcal{P}$ has got infinitely many (reachable) states.

Nonetheless, the problem is decidable!
Finite automata for configurations

To represent (infinite) sets of configurations, we shall employ finite automata. Let \( \mathcal{P} = (P, \Gamma, \Delta) \) be a PDS. We call \( \mathcal{A} = (Q, \Gamma, P, T, F) \) a \( \mathcal{P} \)-automaton.

The alphabet of \( \mathcal{A} \) is the stack alphabet \( \Gamma \).

The initial states of \( \mathcal{A} \) are the control states \( P \).

We say that \( \mathcal{A} \) accepts the configuration \( pw \) if \( \mathcal{A} \) has got a path labelled by input \( w \) starting at \( p \) and ending at some final state.

Remark: In the following, we shall use the following notation:

\[ pw \Rightarrow p'w' \text{ (in the PDS } \mathcal{P} \text{)} \quad \text{and} \quad p \xrightarrow{w} q \text{ (in } \mathcal{P} \text{-automata)} \]
An automaton is **normalized** if there are no transitions leading into initial states. (And any automaton can be brought into a normalized form.)

Let \( \text{pre}^*(C) = \{ c' \mid \exists c \in C : c' \Rightarrow c \} \) denote the predecessors of \( C \).

The following result is due to Büchi (1964):

Let \( C \) be a regular set and \( A \) be a *normalized \( \mathcal{P} \)-automaton* accepting \( C \).

If \( C \) is regular, then so is \( \text{pre}^*(C) \).

Moreover, \( A \) can be transformed into an automaton accepting \( \text{pre}^*(C) \).
The basic idea (for \textit{pre})

**Saturation rule**: Add new transitions to $\mathcal{A}$ as follows:

If $q \xrightarrow{w} r$ currently holds in $\mathcal{A}$ and $pA \leftrightarrow qw$ is a rule, then add the transition $(p, A, r)$ to $\mathcal{A}$.

Repeat this until no other transition can be added.

At the end, the resulting automaton accepts $\textit{pre}^*(C)$.

Complexity: $\mathcal{O}(|Q|^2 \cdot |\Delta|)$ time.
Automaton $A$ for $C$
Extending $\mathcal{A}$

Rule: $pA \rightarrow qB$

Path: $qB \rightarrow s1$

New path: $pA \rightarrow s1$
Extending $\mathcal{A}$

If the right-hand side of a rule can be read,

$$\text{Rule: } pA \leftrightarrow qB \quad \text{Path: } q \xrightarrow{B} s_1$$
Extending $\mathcal{A}$

If the right-hand side of a rule can be read, add the left-hand side.

Rule: $pA \leftrightarrow qB$  \hspace{1cm} Path: $q \xrightarrow{B} s_1$  \hspace{1cm} New path: $p \xrightarrow{A} s_1$
Extending $\mathcal{A}$

If the right-hand side of a rule can be read,

Rule: $pC \leftrightarrow pAD$     Path: $p \xrightarrow{A} s_1 \xrightarrow{D} s_2$
Extending $\mathcal{A}$

If the right-hand side of a rule can be read, add the left-hand side.

Rule: $pC \leftrightarrow pAD$  
Path: $p \xrightarrow{A} s_1 \xrightarrow{D} s_2$  
New path: $p \xrightarrow{C} s_2$
Final result
Proof of correctness

We shall show:

Let $\mathcal{B}$ be the $\mathcal{P}$-automaton arising from $\mathcal{A}$ by applying the saturation rule. Then $\mathcal{L}(\mathcal{B}) = \text{pre}^*(\mathcal{C})$.

Part 1: Termination

The saturation rule can only be applied finitely many times because no states are added and there are only finitely many possible transitions.

Part 2: $\text{pre}^*(\mathcal{C}) \subseteq \mathcal{L}(\mathcal{B})$

Let $c \in \text{pre}^*(\mathcal{C})$ and $c' \in \mathcal{C}$ such that $c'$ is reachable from $c$ in $k$ steps. We proceed by induction on $k$ (simple).
Part 3: $\mathcal{L}(\mathcal{B}) \subseteq \text{pre}^*(\mathcal{C})$

Let $\rightarrow_i$ denote the transition relation of the automaton after the saturation rule has been applied $i$ times.

We show the following, more general property: If $p \xrightarrow{w}_i q$, then there exist $p'w'$ with $p' \xrightarrow{w'}_0 q$ and $pw \Rightarrow p'w'$; if $q \in P$, then additionally $w' = \varepsilon$.

Proof by induction over $i$: The base case $i = 0$ is trivial.

Induction step: Let $t = (p_1, A, q')$ be the transition added in the $i$-th application and $k$ the number of times $t$ occurs in the path $p \xrightarrow{w}_i q$.

Induction over $k$: Trivial for $k = 0$. So let $k > 0$. 


There exist $p_2, p', u, v, w', w_2$ with the following properties:

1. $p \xrightarrow{u} p_1 \xrightarrow{A} q' \xrightarrow{v} q$ (splitting the path $p \xrightarrow{w} q$)
2. $p_1 A \hookrightarrow p_2 w_2$ (pre-condition for saturation rule)
3. $p_2 \xrightarrow{w_2} q'$ (pre-condition for saturation rule)
4. $pu \Rightarrow p_1 \varepsilon$ (ind.hyp. on $i$)
5. $p_2 w_2 v \Rightarrow p' w'$ (ind.hyp. on $k$)
6. $p' \xrightarrow{w'} q$ (ind.hyp. on $k$)

The desired proof follows from (1), (4), (2), and (5).

If $q \in P$, then the second part follows from (6) and the fact that $A$ is normalized.
LTL and Pushdown Systems

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS with initial configuration $c_0$, let $\mathcal{T}_\mathcal{P}$ denote the corresponding transition system, $AP$ a set of atomic propositions, and $\nu: P \times \Gamma^* \rightarrow 2^{AP}$ a valuation function.

$\mathcal{T}_\mathcal{P}$, $AP$, and $\nu$ form a Kripke structure $\mathcal{K}$; let $\phi$ be an LTL formula (over $AP$).

Problem: Does $\mathcal{K} \models \phi$?

Undecidable for arbitrary valuation functions!
(could encode undecidable decision problems in $\nu$ . . .)

However, LTL model checking is decidable for certain restrictions of $\nu$. 
In the following, we consider “simple” valuation functions satisfying the following restriction:

\[ \nu(pAw) = \nu(pA), \text{ for all } p \in P, \ A \in \Gamma, \text{ and } w \in \Gamma^*. \]

In other words, the “head” of a configuration holds all information about atomic propositions.

LTL model checking is decidable for such “simple” valuations.
Approach

Same principle as for finite Kripke structures:

Translate $\neg \phi$ into a Büchi automaton $B$.

Build the cross product of $K$ and $B$.

Test the cross product for emptiness.

Note that the cross product is not a Büchi automaton in this case, but another pushdown system (with a Büchi-style acceptance condition).
The cross product is a new pushdown system $Q$, as follows:

Let $P = (P, \Gamma, \Delta)$ be a PDS, $p_0w_0$ the initial configuration, and $AP, \nu$ as usual.

Let $B = (Q, 2^{AP}, q_0, T, F)$ be the Büchi automaton for $\neg \phi$.

Construction of $Q$:

$Q = (P \times Q, \Gamma, \Delta')$, where

$(p, q)A \mapsto (p', q')w \in \Delta'$ iff

- $pA \mapsto p'w \in \Delta$ and
- $(q, L, q') \in T$ such that $\nu(pA) = L$.

Initial configuration: $(p_0, q_0)w_0$
Let $\rho$ be a run of $Q$ with $\rho(i) = (p_i, q_i)w_i$.

We call $\rho$ accepting if $q_i \in F$ for infinitely many values of $i$.

The following is easy to see:

$\mathcal{P}$ does not satisfy $\phi$ iff there exists an accepting run in $Q$. 

Characterization of accepting runs

Question: If there an accepting run starting at \((p_0, q_0)w_0\)?

In the following, we shall consider the following, more general global model-checking problem:

Compute all configurations \(c\) such that there exists an accepting run starting at \(c\).

Lemma: There is an accepting run starting at \(c\) iff there exists \((p, q) \in P \times Q, A \in \Gamma\) with the following properties:

(1) \(c \Rightarrow (p, q)Aw\) for some \(w \in \Gamma^*\)

(2) \((p, q)A \Rightarrow (p, q)Aw'\) for some \(w' \in \Gamma^*\), where

the path from \((p, q)A\) to \((p, q)Aw'\) contains at least one step;
the path contains at least one accepting Büchi state.
Repeating heads

We call \((p, q)A\) a repeating head if \((p, q)A\) satisfies properties (1) and (2).

Strategy:

1. Compute all repeating heads \((p, q)A\).
   E.g., check for each head if \((p, q)A \in pre^*\{ (p, q)Aw \mid w \in \Gamma^* \}\).
   (Additionally, one needs to check whether an accepting state is visited along the way, which can be encoded into the control state.)

2. Compute the set \(pre^*\{ (p, q)Aw \mid (p, q)A \text{ is a repeating head, } w \in \Gamma^* \}\)
Remarks

Other temporal logics for PDS are also decidable (sketch):

**CTL:** Translate formula into an *alternating* automaton, adapt *pre* algorithm to alternating automata, then apply a technique similar to LTL.

**CTL*: Adapt the technique from finite-state systems: Find an $E$-free subformula $\phi$, compute the (regular) set configurations $C$ satisfying $E\phi$. Then encode the states of the automaton for $C$ into the stack, replace $E\phi$ by a fresh atomic proposition $p$ that is true whenever the modified stack tells us that we are in a configuration satisfying $E\phi$. 