Pushdown systems
Example 1

A small program (where \( n \geq 1 \)):

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

Question: Will \( g \) be true when the program terminates?
Example 1 has got \textit{finitely} many states. 
(The call stack is bounded by \textit{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 \textit{recursive} procedure calls.
Example 2: Recursive program

\begin{align*}
\text{procedure } p; \\
p_0: & \quad \text{if } ? \text{ then} \\
p_1: & \quad \text{call } s; \\
p_2: & \quad \text{if } ? \text{ then call } p; \text{ end if; } \\
\quad \text{else} \\
p_3: & \quad \text{call } p; \\
\quad \text{end if} \\
p_4: & \quad \text{return}
\end{align*}

\begin{align*}
\text{procedure } s; \\
s_0: & \quad \text{if } ? \text{ then return; end if;} \\
s_1: & \quad \text{call } p; \\
s_2: & \quad \text{return}
\end{align*}

\begin{align*}
\text{procedure } \text{main}; \\
m_0: & \quad \text{call } s; \\
m_1: & \quad \text{return}
\end{align*}

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

Inlining not applicable!

Cannot be analyzed by naively searching all reachable states.

We shall require a finite representation of infinitely many states.
Example 3: Quicksort

```c
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)
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 $pA \leftrightarrow qw$, where $p, q \in P$, $A \in \Gamma$, $w \in \Gamma^*$. 

Like acceptors for context-free language, but without any input!
Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS and $c_0 \in P \times \Gamma^*$. 

With $\mathcal{P}$ we associate a transition system $T_\mathcal{P} = (S, \rightarrow, r)$ 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 $pA \leftrightarrow q w \in \Delta$;

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

$pA \leftrightarrow qB$
$pA \leftrightarrow pC$
$qB \leftrightarrow pD$
$pC \leftrightarrow pAD$
$pD \leftrightarrow p\epsilon$
Procedural programs and PDSs

$P$ may represent the valuations of global variables.

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

Interpretation of a configuration $pAw$:

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

Rules:

- $pA \leftrightarrow qB \equiv \text{statement within a procedure}$
- $pA \leftrightarrow qBC \equiv \text{procedure call}$
- $pA \leftrightarrow q\varepsilon \equiv \text{return from a procedure}$
Reachability in PDS

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

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.
Let $\mathcal{L}(A)$ be the set of configurations accepted by $A$.

A set $C$ of configurations is called regular iff there is some $\mathcal{P}$-automaton $A$ with $\mathcal{L}(A) = C$.

An automaton is normalized if there are no transitions leading into initial states.

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

\[ pw \Rightarrow p'w' \text{ (in the PDS } \mathcal{P}) \quad \text{and} \quad p \xrightarrow{w} q \text{ (in } \mathcal{P}\text{-automata)} \]
Reachability in PDS

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

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 are $\text{pre}^*(C)$ and $\text{post}^*(C)$.

Moreover, $A$ can be transformed into an automaton accepting $\text{pre}^*(C)$ resp. $\text{post}^*(C)$. 
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 $p\mathcal{A} \leftrightarrow qw$ is a rule, then add the transition $(p, \mathcal{A}, r)$ to $\mathcal{A}$.

Repeat this until no other transition can be added.

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

For $\text{post}^* (C)$: similar procedure.
Automaton $A$ for $C$
Extending $\mathcal{A}$

Rule: $p \xrightarrow{A} \rightarrow q \xrightarrow{B} s_1$

Path: $q \xrightarrow{B} s_1$

New path: $p \xrightarrow{A} s_1$
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: $pA \leftrightarrow qB$  Path: $q \xrightarrow{B} s_1$  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: $p \xrightarrow{C} s_2$
Complexity: $O(|Q|^2 \cdot |\Delta|)$ time.
Proof of correctness

We shall show:

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

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

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

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 $j$ the number of times $t$ occurs in the path $p \xrightarrow{w}_i q$.

Induction over $j$: Trivial for $j = 0$. So let $j > 0$. 
There exist $p_2, p', u, v, w', w_2$ with the following properties:

(1) $p \xrightarrow{u} p_1 \xrightarrow{A} q' \xrightarrow{v} q$ \hspace{1em} (splitting the path $p \xrightarrow{w} q$)

(2) $p_1 A \hookrightarrow p_2 w_2$ \hspace{1em} (pre-condition for saturation rule)

(3) $p_2 \xrightarrow{w_2} q'$ \hspace{1em} (pre-condition for saturation rule)

(4) $pu \Rightarrow p_1 \varepsilon$ \hspace{1em} (ind.hyp. on $i$)

(5) $p_2 w_2 v \Rightarrow p' w'$ \hspace{1em} (ind.hyp. on $j$)

(6) $p' \xrightarrow{w'} q$ \hspace{1em} (ind.hyp. on $j$)

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

If $q \in \mathcal{P}$, then the second part follows from (6) and the fact that $\mathcal{A}$ is normalized.
Example: \textit{post}^* \ (\textit{without proof})

If the \textit{left-hand side} of a rule can be read,

\begin{center}
\begin{tikzpicture}
  \node [shape=circle,draw=blue] (A) at (0,0) {p};
  \node [shape=circle,draw=blue] (B) at (2,0) {s2};
  \draw [->,blue] (A) -- (B) node [midway, above] {C};
\end{tikzpicture}
\end{center}

Rule: \( pC \leftrightarrow pAD \) \quad Path: \( p \xrightarrow{C} s_2 \)
Example: $post^*$ (without proof)

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

Rule: $pC \iff pAD$  Path: $p \xrightarrow{C} s_2$  New Path: $p \xrightarrow{AD} s_2$
Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS with initial configuration $c_0$, let $T_\mathcal{P}$ denote the corresponding transition system, $AP$ a set of atomic propositions, and $\nu: P \times \Gamma^* \rightarrow 2^{AP}$ a valuation function.

$T_\mathcal{P}$, $AP$, 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 “reasonable” restrictions of $\nu$.  

31
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 $\mathcal{B}$.

Build the cross product of $\mathcal{K}$ and $\mathcal{B}$.

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

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

- $pA \leftrightarrow p'w \in \Delta$ 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:

$P$ 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.
   E.g., check for each pair \((p, q)A\) whether \((p, q)A \in \text{pre}^*\{ (p, q)Aw \mid w \in \Gamma^* \}\). Visiting an accepting state can be encoded into the control state. (This is a simple but naïve method, one can do better.)

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