Supplemental material taken from Lecture 7 of a set of lecture presentations by Mooly Sagiv of Tel-Aviv University. Based on Section 2.6 of the text "Principles of Program Analysis."
Abstract Syntax

```plaintext
S ::= x := Aexp | x : sel := Aexp | x := malloc() | skip | if Bexp then S1 else S2
S1 := while Bexp do S1 | S2
Aexp ::= x | x : sel | Aexp := x . sel | Aexp := : x
Bexp ::= null | 0 | I | Aexp | Bexp op Aexp | Aexp rel Aexp
Aexp ::= malloc() | sel | car | cdr | val
```

We introduce the SWhile programming language, which includes dynamic memory allocation, pointers, and destructive updates. We will define operational semantics to this language, using the notion of a fair discrete systems (FDS).
A fair discrete system (FDS) consists of:

\[ \langle \mathcal{C}, \mathcal{D}, d, \Theta, \Lambda \rangle = FDS \]
Computations

Let \( D \) be an FDS for which the above components have been identified. The state \( s_0 \) is defined to be a \( D \)-successor of state \( s \) if

\[
\Theta \models s_0 \rightarrow \omega_s \neq 0.
\]

We define a computation of \( D \) to be an infinite sequence of states

\[
(\omega_A, \Lambda) \models D = \langle s', s \rangle.
\]

Let \( D \) be an FDS for which the above components have been identified. The state \( s \) is defined to be a \( D \)-successor of state \( s_0 \) if

\[
\Theta \models s_0 \rightarrow \omega_s \neq 0.
\]

must also contain infinitely many \( b \)-positions.

Compassion: For each \( \rho \in \mathcal{C} \), \( \rho \) contains infinitely many \( p \)-positions.

Justice: For each \( j \in \mathcal{J} \), \( \rho \) contains infinitely many \( j \)-positions.

Consecution: For each \( s \in \mathcal{S} \), \( \rho \) contains infinitely many \( \rho \)-positions.

Initiality: \( s_0 \) is initial, i.e., \( s_0 \models \Theta \).

Satisfying the following requirements:

\[
\begin{align*}
\rho & : s_0, s_1, s_2, \ldots, \\
& \vdash \Theta.
\end{align*}
\]
The domain of car or cdr belonging to \( O \) is required that all the objects appearing as values of program variables or in the domain of car, cdr, or null be atoms, objects, or null.

Program variables assinging values which could be atoms, objects, or null:

\[ \{ \text{null} \} \cap I \cap D : x_1, \ldots, x_n \]

null: car: \[ \{ \text{null} \} \cap I \cap D \leftarrow I \]

cdr: \[ \{ \text{null} \} \cap I \cap D \leftarrow I \]

null: \( O \) — the set of allocated objects. It ranges over finite subsets of \( I \).

executed: \( \neg \) — the program counter pointing to the label of the next statement to be executed.

The structure of SWhile states contains the following:

SWhile program, the state variables will include the following:

and a universal set of (memory) locations (also called objects). For every \( I \) and a universal set of data values (also called atoms) (integers, booleans, etc.).

We assume a set of data values (also called atoms) \( D \)
Example: Program REVERSE

Consider program REVERSE:

\[
\begin{align*}
&\text{return } y \\
&\text{while } x \neq \text{null} \\
&\text{null} =: y \\
&\text{null} \neq x \\
&\text{null} =: y \\
&(\text{list } \text{reverse}(\text{list } x))
\end{align*}
\]

This program may give rise to the execution state described by:

\[
\begin{bmatrix}
\begin{array}{ccc}
\ell =: \text{cdr}.y & x =: x & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{array}
\end{bmatrix}
\]

Example: Program Analysis
The state can be represented as follows:

![Diagram showing state representation]

null : \( \top \)
null : \( \bot \)
\( x \) : \( x \)
\( o_1 \) : \( o_1 \)
\( o_2 \) : \( o_2 \)
\( o_3 \) : \( o_3 \)

The state

Representation of the State

A. Pnueli

Program Analysis, NYU, Spring, 2004
Transition Relation and Justice

For each type of statement, we indicate the disjunct contributed to the transition relation \( \text{at}(\cdot) \), the justice, and the compassion requirements contributed by the statement. Then the notations \( \text{at}(\cdot) \) and \( \text{at}(\cdot) \) respectively.

\[
\{f_1, \ldots, f_n\} \cdot \Lambda \overset{\text{pres}}{\supset} \forall \ e = f_i \land (\forall)_{\text{at}} (\forall)_{\text{at}} \land \\text{at} \quad \text{at}(\cdot)
\]

 Consider the statement \( \{f_1, \ldots, f_n\} \cdot \Lambda \overset{\text{pres}}{\supset} \forall \ e = f_i \land (\forall)_{\text{at}} (\forall)_{\text{at}} \land \\text{at}(\cdot) \).

We use the notation \( \text{pres}(\cdot) \) as an abbreviation for the assignment statement.

\[
\forall \ e = f_i \land (\forall)_{\text{at}} (\forall)_{\text{at}} \land \\text{at}(\cdot) \supset (\forall)_{\text{pres}} (\forall)_{\text{pres}} (\forall)_{\text{pres}} (\forall)_{\text{pres}}
\]

The assignment statement \( \{f_1, \ldots, f_n\} \cdot \Lambda \overset{\text{pres}}{\supset} \forall \ e = f_i \land (\forall)_{\text{at}} (\forall)_{\text{at}} \land \\text{at}(\cdot) \) contributes to the disjunct \( \text{at}(\cdot) \) and \( \text{at}(\cdot) \) respectively.

The assignment statement \( \{f_1, \ldots, f_n\} \cdot \Lambda \overset{\text{pres}}{\supset} \forall \ e = f_i \land (\forall)_{\text{at}} (\forall)_{\text{at}} \land \\text{at}(\cdot) \) and the compassion requirements contributed by the statement.

We use the notation \( \text{pres}(\cdot) \) as an abbreviation for the assignment statement.

\[
\forall \ e = f_i \land (\forall)_{\text{at}} (\forall)_{\text{at}} \land \\text{at}(\cdot) \supset (\forall)_{\text{pres}} (\forall)_{\text{pres}} (\forall)_{\text{pres}} (\forall)_{\text{pres}}
\]
The assignment \( j : y := e \); \( k \) contributes to the disjunct \( \text{at}(t_0 \land \text{at}(0) \land o) \). For example, executing statement \( 6: y \text{: cdr} := t; 2 : \text{in the state:} \)

\[
\begin{array}{ccc}
\text{null} & \text{null} & \text{null} \\
\text{x} & \text{z} & \text{t} \\
\text{null} & \text{null} & \text{null} \\
\end{array}
\]

yields the state

\[
\begin{array}{ccc}
\text{null} & \text{null} & \text{null} \\
\text{x} & \text{z} & \text{t} \\
\text{null} & \text{null} & \text{null} \\
\end{array}
\]

and contributes to the requirement \( L \). The assignment \( j : y \text{: sel} := \text{null}; e \text{: sel} := \text{null} \) contributes to the disjunct \( d \).

**Transition Relation 2**
and contributes to the requirement $L$.

\[(\{u\} - \Lambda)\text{pre} \lor \left( (\forall)_{\text{at}} \lor q \right) \land (\forall)_{\text{at}} \lor q \right) \lor (\forall)_{\text{at}}
\]

- The while statement $f$: while $q$ do $S$ \[
\text{contributes to the disjunct $d$}
\]

\[(\{u\} - \Lambda)\text{pre} \lor \left( (\forall)_{\text{at}} \lor q \right) \land (\forall)_{\text{at}} \lor q \right) \lor (\forall)_{\text{at}}
\]

- The conditional statement $f$: if $q$ then $S_1$ else $S_2$ \[
\text{contributes to the disjunct $d$}
\]

\[(\{\tilde{u}\} - \Lambda)\text{pre} \lor \left( (\forall)_{\text{at}} \lor q \right) \land (\forall)_{\text{at}} \lor q \right) \lor (\forall)_{\text{at}}
\]

- The assignment statement $f$: \[
\text{contributes to the disjunct $d$}
\]

**Transition Relations 3**
For the minimal set of distinguishing predicates, we consider the predicate $\forall n: x^n \leq x^n$.

Trying to mechanically analyze even a program as simple as \textsc{Reverse} is impossible because the generated data structures (shapes) are unbounded.
Shapes Arising In List Reversal

Stage 5:

Stage 4:

Stage 3:

Stage 2:

Stage 1:

Stage 0:

Shapes Arising In List Reversal