Next, we will consider methods for proving response properties under general fairness. This means that, in addition to justice requirements, we should also consider the presence of compassion requirements. Next, we will consider methods for proving response properties under general fairness.
Recall the two types of fairness that were traditionally introduced into the formal model of reactive systems:

Two Types of Fairness

A. Pnueli

Deductive Verification of Reactive Systems, NYU, Fall, 2007
In the Model of Fair Discrete Systems (FDS)
Answer: Justice is Easy, Compassion is Complex

This used to be the consensus, manifested in model checking and deductive verification. In this presentation, we claim that this is not necessarily the case, and illustrate it on the process of deductive verification.
Verifying Response Under Justice

Rule J-WELL

For a well-founded domain \((A, \prec)\)

and ranking functions

Justice requirements

assertions

Easy!!!

Deductive Verification of Reactive Systems, NYU, Fall, 2007
Lecture 9: Liveness under Compassion

A. Pnueli

Response Under General Fairness (Legacy)

Rule F-WELL [MP91]

\[ b \triangleleft d \]

F. Pnueli

\[ (\forall \rho . \rho \wedge \rho') \equiv (\forall \rho . \rho) \wedge (\forall \rho . \rho') \wedge b \leftarrow d \wedge \rho \]

F. 3.

\[ \rho \wedge b \leftarrow d \vee \rho \]

F. 2.

\[ \rho' \wedge b \leftarrow \rho \]

F. 1.

\[ \forall \rho \leftarrow \exists \rho' : \rho \wedge \rho' \wedge b \leftarrow d \]

and ranking functions \[ \rho, \rho' \] where each \[ \rho \subset \rho' \] and ranking functions \[ \rho, \rho' \] where each \[ \rho \subset \rho' \]

Fairness requirements \[ \forall \rho, \rho' \exists \rho'' \rho \wedge \rho' \wedge \rho'' \wedge \rho \]

For a well-founded domain \[ d \in (W, \prec) \]
Represent justice as degenerate compassion.

\[
\begin{align*}
\forall & \iff \exists : \forall q, \ldots, q_1, (u_1, u_d) \ldots, (u, u_d) (\forall, <) \\
\text{(assertion requirements)} & \text{and ranking functions for a well-founded domain}\n\end{align*}
\]

\[
\begin{align*}
& b \Diamond \iff d \\
& \vdash b \iff \forall q \land \forall \forall y \land \forall d \land \forall u \land \forall v \land \forall w (q \lor y \lor d) \land (q = q \lor y) \land b \iff d \lor y \\
&W_3.
& \forall \forall q \land \forall d \land \forall u \land \forall v \land \forall w (q \lor y \lor d) \land (q \lor y) \iff \forall q \land \forall d \\
&W_2.
& \forall \forall q \land \forall d \land \forall u \land \forall v \land \forall w (q \lor y \lor d) \land (q \lor y) \iff \forall q \land \forall d \\
&W_1.
& \forall \forall q \land \forall d \land \forall u \land \forall v \land \forall w (q \lor y \lor d) \land (q \lor y) \iff \forall q \land \forall d \\
&W_4.
& \forall \forall q \land \forall d \land \forall u \land \forall v \land \forall w (q \lor y \lor d) \land (q \lor y) \iff \forall q \land \forall d \\
&W_5.
\end{align*}
\]

So What's New? A Flat Rule for Compassion
Consider the following program: 

```
COND - TERM

\[
x, y : \text{natural init } x = 0
\]

\[
l_1 : \text{while } y > 0 \text{ do}
\]

\[
\begin{align*}
  & l_2 : x := \{0, 1\} \\
  & l_3 : y := y + 1 - 2x \\
  & l_4 : \text{end}
\end{align*}
\]

Prove the response property at \( l_1 \) under the compassion requirement (at \( l_3 \land x = 0 \), implying that \( x \) can assume the value 0 only finitely many times.

Consider the following rank-abstracted version of this program:

```
X, Y : \{0, 1\} \text{ init } X = 0

\[
l_1 : \text{while } Y = 1 \text{ do}
\]

\[
\begin{align*}
  & l_2 : X := \{0, 1\} \\
  & l_3 : \begin{cases}
    (1, 1) & \text{if } X = 0 \\
    (1, -1), (0, -1) & \text{else}
  \end{cases} \\
  & l_4 : \text{end}
\end{align*}
\]
Lecture 9: Liveness under Compassion

A. Pnueli

Model Checking + Helpful Assertions Extraction

Wishing to verify \( \text{init} \) stability

\[ 0 =: p \]
\[ \text{States reachable from an init-state by a } g\text{-free path.} \]

If then property is valid and the auxiliary constructs provide a deductive proof.

\[ \emptyset = X \]
\[ (\|d\| \cup C) - X =: X \]
\[ \|C\| =: p q \quad (b, d) =: p \quad I + p =: p \]

5. Contains a \( b\)-state but no \( b\)-state.

4. Otherwise, \( C \) is compassionate requirement construct a counterexample.

3. If \( C \) is compassionate — construct a counterexample.

2. Let \( C \) be a terminal MSCC.

1. Decompose \( X \) into MSCCs.

Repeat until \( X \) stabilizes

\[ 0 =: p \]
\[ \text{States reachable from an init-state by a } g\text{-free path.} \]

Wishing to verify \( \text{init} \) stability

\[ 0 =: p \]
\[ \text{States reachable from an init-state by a } g\text{-free path.} \]
Pending States
Removing $K_1$
Removing $K_2$

$l_1, Y: 1, D: 1$

$l_2$

$l_3, Y: 1, X: 1$

$F_3: (D < 0, D > 0)$

$l_1, Y: 1, D: -1$
Removing $K^3_3$
Removing $K_5$
To prove termination of the original (infinite-state) system, using rule C-Well, we choose the following constructs for \( n = 4 \):

<table>
<thead>
<tr>
<th>((1, 2))</th>
<th>((1, 3))</th>
<th>((1, 4))</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1, 1))</td>
<td>((1, 2 - \text{alt}))</td>
<td>((1, \text{alt}))</td>
</tr>
<tr>
<td>((0, 1))</td>
<td>((0, 1))</td>
<td>((0, 0))</td>
</tr>
<tr>
<td>((0, 0 = x \lor 3 - \text{alt}))</td>
<td>{(0, 0) \in x \lor 3 - \text{alt}} \subseteq x \lor 3 - \text{alt})</td>
<td>(x \lor 3 - \text{alt})</td>
</tr>
<tr>
<td>(0)</td>
<td>(d^4)</td>
<td>(d)</td>
</tr>
</tbody>
</table>

Deductive Proof of COND-TERM
Consider the following program: 

```
local y : boolean
init y = 1
i = 1

loop forever do

  `1: NonCritical
  request y

  `2: Critical
  release y

  `3: request y

  `4: release y

loop forever do

  `1: boolean init
  i = y

  `2: Local
```

We wish to prove the response property property at `2; `3.

To use rule C-WELL, we choose the following constructs:

```
[ ? ] a \arrow [ ? ] a 
```

We also use the following auxiliary invariant:

\[ I = [?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u}) \]

Example: Program MUX-SEM:

```
A. Pnueli

Example: Program MUX-SEM

\[ \begin{array}{c|c|c|c}
\hline
2 & ([?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u})) & ([?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u})) & I < \mathbf{c}(\mathbf{f}, \mathbf{e}) \\
1 & ([?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u})) & ([?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u})) & I < \mathbf{c}(\mathbf{f}, \mathbf{e}) \\
0 & ([?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u})) & ([?](a \rightarrow [?](a \land \neg \exists \mathbf{u} \neg I = y) + y : \not\in I = \mathbf{u})) & I \\
\hline
\end{array} \]

```

Deductive Verification of Reactive Systems, NYU, Fall, 2007
In the file `muxsem.smv`, we place:

```
LECTURE 9: Liveness under Compassion

MODULE main
DEFINEN:=6; loc1:=P[1].loc;
VAR y:boolean;
P: array1..N of process MP(y);
MODULE MP(y)
VAR loc: 1..4;
ASSIGN init(loc):=1; init(y):=1;
next(loc):=case loc=1:{1,2}; loc=2&y:3; loc=3:4; loc=4:1; esac;
next(y):=case loc=2&y:0; loc=4:1; 1:y; esac;

JUSTICE loc!=3, loc!=4
COMPASSION (loc=2 & y, loc = 2)
Liveness (loc = 3, loc = 4)
```

```
Doing it in TLV
```
Let prepare.

In file `muxsem.pf`, we place:

The Proof Script

```
A. Pnueli

Let "h", "fairp", "fairq", and Del are reserved variables.

Let Del[nas] := 2

Let Del[nas][1] := p[j].loc = 3

Let Del[nas] := 1

Let h[nas] := loc1=2 & p[j].loc = 3

Let h[nas] := nas + 1

For j in 2..N

Let nas := 1

Let Del[nas] := 0

Let Del[nas][1] := p[j].loc = 3

Let h[nas] := loc1=2 & p[j].loc = 3

Let Del[nas][1] := 2

End -- For j in 2..N

Let h[nas] := 0

For j in 1..N

Let h[nas] := p[j].loc = 3

Let Del[nas][j] := nas + 1

Let h[nas] := loc1=2 & p[j].loc = 3

Let Del[nas][j] := 1

End -- For j in 1..N

Let sum := 0

For j in 1..N

Let Del[nas][j] := nas + 1

Let h[nas] := loc1=2 & p[j].loc = 3

End -- For j in 1..N

Let sum := sum + p[j].loc = 3

Let Del[nas][j] := 0

End -- For j in 1..N

Let sum := sum

Let h[nas] := loc1=2 & p[j].loc = 3

Let Del[nas][j] := 1

End -- For j in 1..N

```

Deductive Verification of Reactive Systems, NYU, Fall 2007
Lecture 9: Liveness under Compassion

A. Pnueli

Deductive Verification of Reactive Systems, NYU, Fall 2007

Let \( n_{as} := n_{as} + 1 \);

Let \( n := 2 \land P[j].loc = 4 \);

Let \( fair_p[n_{as}] := 1 \);

Let \( fair_q[n_{as}] := P[j].loc \neq 4 \);

Let \( Del[n_{as}][1] := 1 \);

End For (j in 2..N)

End Toprepare;

To compute;

Print "\nCheck invariance of _inv\n";

Call.binv(_inv, 1);

Print "\nModel check response property\n";

Call.Temp.Entail(start, goal, 1);

Print "\nDeductively check response property\n";

Let result := flat_response(start, goal, _inv, nas, 1, 1);

If (result)

Print "\n***Response property is valid***\n";

End If (result)

End Toprepare;

End -- To compute!

End -- If (result)

Let result := flat_response(start, goal, _inv, nas, 1, 1, 1);

Print "\nIn deductively check response property\n";

Call.Temp.Entail(start, goal, 1);

Print "\nIn Model check response property\n";

Call.binv(_inv, 1);

Print "\nIn check invariance of _inv\n";

To compute;

End -- To prepare;

End -- For (j in 2..N)

Let Del[nas][1] := 1;

Let fair_p[nas] := 1;

Let fair_q[nas] := P[j].loc = 4;

Let Inv[nas] := loc1 = 2 \land p[j].loc = 4;

Let nas := nas + 1;
If follows that for all $\phi, \psi$:

\[(\mathcal{W} = (1+fs)wq) \lor (1+fs)wy \iff (\mathcal{W} = (fs)wq) \lor (fs)wy\]

Since never holds beyond $a$ and is minimal, this can be simplified to:

\[((1+fs)fq \land (1+fs)f\psi) \land (a = f \land (\mathcal{W} = (1+fs)wq) \lor (1+fs)wy) \iff (\mathcal{W} = (fs)wq) \lor (fs)wy\]

According to W4, let $a > c$ and $m \in \mathcal{W}$ be such that $[u\ldots]$ be such that $\mathcal{W}$ is minimal. Let $\mathcal{W}$ be such a minimal value in $\text{Rank-Set}$. Then, $\mathcal{W}$ holds at infinitely many positions in $\mathcal{W}$.

There exists $c \geq f$, such that for every $c \leq f$, if $s \in \mathcal{W}$, then $\mathcal{W} \lor s = \mathcal{W}$. If $s \in \mathcal{W}$, then $\mathcal{W} \lor s = \mathcal{W}$. If $s \in \mathcal{W}$, then $\mathcal{W} \lor s = \mathcal{W}$.

By premise W2, for every $f \in [u\ldots]$ there exists at least $f$ many positions in $\mathcal{W}$.

From premises W1 and W3, it follows that for all $\phi$ and $\psi$:

\[\phi \lor \psi \iff \phi \lor (\mathcal{W} \lor \psi)\]

Since never holds beyond $a$ and is minimal, this can be simplified to:

\[\mathcal{W} = (fs)wq \lor \mathcal{W} \lor \psi = (fs)wq \lor \mathcal{W} \lor \psi\]

We show that if the premises of rule C-WELL are valid, then so is the conclusion.

**Rule C-WELL is Sound**
Conclude that $b \diamond \iff d$ is valid.

and cannot, therefore, be a computation.

The last two items show that $\tau$ violates the requirement of compassion.

positions in $\tau$.

Since $\tau$ and $s$.

By premise $W5$, for all $s$.

Established that $\tau$.
We show that if a finite-state Compassionate discrete system (CDS) satisfies

\[ b \diamond \leftarrow d \]

Rule C-WELL is Complete
Example: MUX-SEM

First step in ranking extraction:

Main compassion requirement requirement

[\text{Example}]:

\[
0 = \bar{y} \lor \left[ \square \neg \neg \square \neg \right] X \lor \left[ \square \neg \square \square \neg \right] X \lor \left( \left[ \square \neg \neg \right] Y \lor \left[ \square \neg \square \neg \right] Y \right) \\downarrow \frac{0}{} \frac{1}{} \frac{1}{} \frac{1}{}
\]

Deductive Verification of Reactive Systems, NYU, Fall, 2007

A. Pnueli
Example: MUX-SEM

where \( I < J \) and \( J = 5 - I \).

Next step in ranking extraction:

\[
\begin{array}{c|c|c|c}
\text{MUX-SEM} & \text{al} & \text{at} & \text{at} \\
\text{[I] at} & \text{[I] at} & \text{[I] at} & \text{[I] at} \\
p_h & p_h & p_h & p_h \\
\text{[I] at} & \text{[I] at} & \text{[I] at} & \text{[I] at} \\
\end{array}
\]
Example: MUX-SEM

next step in ranking extraction:

\[ \text{where } \frac{I}{J} < 1 \text{ and } \frac{I}{J} = 5 - \ell \cdot \]

\[
\begin{array}{|c|c|c|c|}
\hline
[\ell]^{I,J} \quad \text{at} \quad [\ell]^{I,J} \quad \text{at} \quad [\ell]^{I,J} \\
[\ell]^{I,J} \quad \text{at} \quad [\ell]^{I,J} \quad \text{at} \quad [\ell]^{I,J} \\
\hline
[1]^{I,J} \quad \text{at} \quad 1 = \bar{\nu} \\
\hline
\end{array}
\]

\[ p_\bar{\nu} \quad p_K \quad \bar{p}_H \quad \bar{p} \]
Consider the following program DINE-PHIL:

\[
\text{local } f : \text{array}[1..n] \text{ of boolean} \text{ init } f = 1
\]

\[
\text{loop forever do}
\]

\[
0 : \text{NonCritical}
\]

\[
1 : \text{request } f[i]
\]

\[
2 : \text{release } f[i]
\]

\[
3 : \text{Critical}
\]

\[
4 : \text{release } f[i+1]
\]

\[
5 : \text{request } f[i]
\]

\[
6 : \text{release } f[i+1]
\]

\[
\text{loop forever do}
\]

\[
1 : \text{request } f[1]
\]

\[
2 : \text{release } f[n]
\]

\[
3 : \text{Critical}
\]

\[
4 : \text{release } f[1]
\]

\[
5 : \text{release } f[n]
\]

For this program we wish to prove the response property:

\[
\text{at } t_3 \gg \Diamond \left[ p = 2 \right] \iff \text{at } t_2 \gg \Diamond \left[ p = 2 \right]
\]
In order to enumerate the fairness requirements (as well as transitions) for this system, we use the indexing scheme \((j; k)\), where \(j\) ranges over processes and \(k\) ranges over locations. To prove this property, using rule \(C\)-WELL, we choose the following constructs:

\[
I = [I]f + [u]\forall j \neq at \land \dot I = [I]f + [I + u]\forall j \neq at + [I - u]\forall j \neq at
\]

\[
\forall (I = [I + u]f + [I + u]\forall j \neq at \land \dot I = [I - u]\forall j \neq at) \lor
\]

We also use the following auxiliary invariant:

<table>
<thead>
<tr>
<th>(y - 3, I)'s (I)'s</th>
<th>(I)'s (I)'s</th>
<th>(I)'s (I)'s</th>
<th>(I)'s (I)'s</th>
</tr>
</thead>
<tbody>
<tr>
<td>((y - 3, I))'s (I)'s</td>
<td>([I]y \neq at \land \dot I = [I]y \neq at)</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
</tr>
<tr>
<td>(0, 0)'s (0, 0)'s</td>
<td>(\forall j \neq at \land \dot I = \forall j \neq at \land \dot I = \forall j \neq at)</td>
<td>([u]y \neq at \land \dot I = [u]y \neq at \land \dot I = [u]y \neq at)</td>
<td>([u]y \neq at \land \dot I = [u]y \neq at \land \dot I = [u]y \neq at)</td>
</tr>
<tr>
<td>(y - 3, I)'s (I)'s</td>
<td>([I]y \neq at \land \dot I = [I]y \neq at)</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
</tr>
<tr>
<td>(0)'s (0)'s</td>
<td>(\forall j \neq at \land \dot I = \forall j \neq at \land \dot I = \forall j \neq at)</td>
<td>([u]y \neq at \land \dot I = [u]y \neq at \land \dot I = [u]y \neq at)</td>
<td>([u]y \neq at \land \dot I = [u]y \neq at \land \dot I = [u]y \neq at)</td>
</tr>
<tr>
<td>((y - 3, I))'s (I)'s</td>
<td>([I]y \neq at \land \dot I = [I]y \neq at)</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
</tr>
<tr>
<td>(0)'s (0)'s</td>
<td>(\forall j \neq at \land \dot I = \forall j \neq at \land \dot I = \forall j \neq at)</td>
<td>([u]y \neq at \land \dot I = [u]y \neq at \land \dot I = [u]y \neq at)</td>
<td>([u]y \neq at \land \dot I = [u]y \neq at \land \dot I = [u]y \neq at)</td>
</tr>
<tr>
<td>((y, I))'s ((y, I))'s</td>
<td>([I, y] \neq at \land \dot I = [I, y] \neq at)</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
</tr>
<tr>
<td>((y, I))'s ((y, I))'s</td>
<td>([I, y] \neq at \land \dot I = [I, y] \neq at)</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
<td>(\text{at} \land \dot I = \text{at} \land \dot I = \text{at})</td>
</tr>
</tbody>
</table>

In order to enumerate the fairness requirements (as well as transitions) for this system, we use the indexing scheme \((j; k)\), where \(j\) ranges over processes and \(k\) ranges over locations. To prove this property, using rule \(C\)-WELL, we choose the following constructs:

\[
I = [I]f + [u]\forall j \neq at \land \dot I = [I]f + [I + u]\forall j \neq at + [I - u]\forall j \neq at
\]

\[
\forall (I = [I + u]f + [I + u]\forall j \neq at \land \dot I = [I - u]\forall j \neq at) \lor
\]

We also use the following auxiliary invariant:

\[
\forall j \neq at \land \dot I = \forall j \neq at \land \dot I = \forall j \neq at
\]