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.

In the Model of Fair Discrete Systems (FDS),

Two Types of Fairness

1. Justice: Justice is Easy, Compassion is Complex

- Guarantee that a computation on a shared resource is achievable in a fair way.
- If transition \( f \) is initially often enabled, it must be taken infinitely many times.

Strong Fairness (Compassion):

- Guarantee that every concurrent process will eventually progress.
- Elected.
- Equivalently, there must be infinitely many states in which \( f \) is disabled.
- If transition \( f \) is continuously enabled, it must be taken infinitely many times.

Weak Fairness (Justice):

- Guarantee that a computation on a shared resource is achievable in a fair way.
- If transition \( f \) is initially often enabled, it must be taken infinitely many times.

Question: Why do we need both types of a compassion requirement? would not be achieved as a special case

Optionally, infinitely many \( p_i \)-states imply infinitely many \( q_i \)-states for each transition \( f \) a set of compassion (strong fairness) requirements.

\[
\{q_1, q_2, \ldots, q_n\} = \emptyset
\]

This has been abstracted as follows:

In the Model of Fair Discrete Systems (FDS),

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.
Consider the following program: 

```
(0 < q > 0) \land (0 = x \lor x) \implies (y = q) \land b
```

Prove the response property. 

**Example Program COND-TERM**

```
Example: Program COND-TERM

<table>
<thead>
<tr>
<th>b \diamond</th>
<th>\iff</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>\iff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\forall y, x \mid \exists x \in D) \rightarrow (y = q) \land b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\forall y, x \mid \exists x \in D) \rightarrow (y = q) \land b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Complex, Recursive**

Representation as degenerate composition:

```
\begin{array}{l}
\forall y, x \mid \exists x \in D \\
\exists x \in D \\
\end{array}
```

**Easy**

```
\begin{array}{l}
\forall y, x \mid \exists x \in D \\
\exists x \in D \\
\end{array}
```

**So What's New? A Flat Rule for Computation**

```
\begin{array}{l}
\forall y, x \mid \exists x \in D \\
\exists x \in D \\
\end{array}
```
Wishing to verify \( \text{init} \rightarrow_{g} g \). 

\[ \begin{align*}
X & := \text{States reachable from an init-state by a } g \text{-free path.} \\
\text{d} & := 0 \\
\text{Repeat until } X \text{ stabilizes} \\
1. \text{Decompose } X \text{ into MSCCs.} \\
2. \text{Let } C \text{ be a terminal MSCC.} \\
3. \text{If } C \text{ is compassionate — construct a counterexample.} \\
4. \text{Otherwise, } (p; q) \in C \text{ s.t. } C \text{ contains a } p \text{-state but no } q \text{-state.} \\
5. \text{d} := \text{d} + 1; F_d := (p; q); h_d := k_C k. \\
6. X := X \setminus C_p k \\
\text{If } X = \emptyset \text{ then property is valid and the auxiliary constructs provide a deductive proof.} \\
\end{align*} \]
Removing $K_3$: $D_3 < D_0; D_0 = 1; Y_1 : 1; D_1 = 1; Y_1 : 1; X_1 : 1; F_4 : (1 < ; at $D_3$)

Removing $K_4$: $D_4 < D_1; D_1 = 1; Y_1 : 1; D_1 = 1; F_5 : (1 < ; at $D_4$)

Removing $K_5$: $D_5 < D_1; D_1 = 1; F_6 : (1 < ; at $D_5$)
To prove termination of the original (infinite-state) system, using rule C-WELL, we choose the following auxiliary invariant:

\[
\begin{align*}
\forall i: P[i] &\vdash (y; 0 < i) \\
\forall i: P[i] &\vdash (y; i > 0)
\end{align*}
\]

We also use the following transition rules (for TLV):

1. \( (\delta, 0) : y \rightarrow (\delta', 1) \)
2. \( (\delta, 1) : y \rightarrow (\delta', 0) \)
3. \( (\delta, 2) : y \rightarrow (\delta', 1) \)
4. \( (\delta, 3) : y \rightarrow (\delta', 0) \)
5. \( (\delta, 0) : y \rightarrow (\delta', 1) \)
6. \( (\delta, 1) : y \rightarrow (\delta', 0) \)
7. \( (\delta, 2) : y \rightarrow (\delta', 1) \)
8. \( (\delta, 3) : y \rightarrow (\delta', 0) \)

Consider the following program MUX-SEM:

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

VAR boolean
DEFINEN:=6; loc1:=P[1].loc;
```

Example: Program MUX-SEM:

Putting it All Together

Deductive Verification of Reactive Systems, NYU, Fall, 2007

Deductive Proof of COMP-TERM

We prove the following transition rules (for TLV):

1. \( (\delta, 0) : y \rightarrow (\delta', 1) \)
2. \( (\delta, 1) : y \rightarrow (\delta', 0) \)
3. \( (\delta, 2) : y \rightarrow (\delta', 1) \)
4. \( (\delta, 3) : y \rightarrow (\delta', 0) \)
5. \( (\delta, 0) : y \rightarrow (\delta', 1) \)
6. \( (\delta, 1) : y \rightarrow (\delta', 0) \)
7. \( (\delta, 2) : y \rightarrow (\delta', 1) \)
8. \( (\delta, 3) : y \rightarrow (\delta', 0) \)

Choose the following transition rules (for TLV):

\[
\begin{align*}
(\delta, 0) &\rightarrow (\delta', 1) \\
(\delta, 1) &\rightarrow (\delta', 0) \\
(\delta, 2) &\rightarrow (\delta', 1) \\
(\delta, 3) &\rightarrow (\delta', 0)
\end{align*}
\]

To prove termination of the original (infinite-state) system, using rule C-WELL, we choose the following auxiliary invariant:

\[
\begin{align*}
\forall i: P[i] &\vdash (y; 0 < i) \\
\forall i: P[i] &\vdash (y; i > 0)
\end{align*}
\]
A.Pnueli

Liveness under
Compassion

Lecture
232

Soundness Proof Continued

Theorem 3.10 (Soundness) For all 

\[ \forall b \in F \text{, } \forall p \in \{\text{fair}\} \text{, if } p \models b \text{ then } p \models \text{fair}. \]

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

Rule C-WELL is Sound

\[ A.Pnueli \]

Liveness under
Compassion

Lecture
233

The Proof Script

-- Note that h, t, capi, capj, and del are reserved variables.

let del [\n  1] := 2;

let capi [\n  ] := 1;

let capj [\n  ] := P[j].loc != 3;

let inv := (sum = 1);

let start := loc1 = 2; let goal := loc1 = 3;

let inv := (sum = 1);

To prepare:

```
Tocompute;
For(i in 1...N)
End--To prepare;
```

```
let nas := nas + 1;

let start := loc1 = 2; let goal := loc1 = 3;
```

```
For(j in 2...N)
End--To prepare;
```

```
let capj [\n  ] := P[j].loc != 4;

let inv := (sum = 1);
```

```
Tocompute;
For(i in 1...N)
End--To prepare;
```

```
let nas := nas + 1;
```

```
let f := [1][\n  ];
```

```
let capi [\n  ] := 1;
```

```
let capj [\n  ] := P[j].loc = 4;
```

```
callbinv(_inv, 1);
```

```
end -- To prepare;
```

Theorem 3.9 (Liveness Under Compassion).

Let Del \[nas\] := 2;

Let fairp \[nas\] := 1;

Let fairq \[nas\] := P[j].loc != 3;

End--To prepare;

```
Let inv := (sum = 1);
```

```
CallTemp_Entail(start, goal, 1);
```

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

```
end -- To prepare;
```

```
Tocompute;
For(i in 1...N)
End--To prepare;
```

```
let inv := (sum = 1);
```

```
End--If(result)
```

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

```
end -- To prepare;
```

```
Tocompute;
For(i in 1...N)
End--To prepare;
```

```
let inv := (sum = 1);
```

```
End--If(result)
```

\[ \forall b \in F \text{, } \forall p \in \{\text{fair}\} \text{, if } p \models b \text{ then } p \models \text{fair}. \]

\[ \forall b \in F \text{, } \forall p \in \{\text{fair}\} \text{, if } p \models b \text{ then } p \models \text{fair}. \]
The page contains a diagram and text discussing the concept of Compassionate Discrete Systems (CDS). The text mentions the use of deduction in verifying properties of these systems and introduces the Rule-CWELL, which is claimed to be complete. The diagram illustrates the ranking extraction process, showing how to calculate the reachability of states in a CDS. The text also mentions the use of MindX-SEMs in this context.
We also use the following auxiliary invariant:

<table>
<thead>
<tr>
<th>(\gamma = 2)</th>
<th>(\gamma = 2)</th>
<th>(\gamma = 2)</th>
<th>(\gamma = 2)</th>
<th>(\gamma = 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>[\gamma]j \rightarrow [\gamma]j \rightarrow \emptyset</td>
<td>\langle [\gamma]j \rightarrow [\gamma]j \rightarrow \emptyset</td>
<td>\langle [\gamma]j \rightarrow [\gamma]j \rightarrow \emptyset</td>
<td>\langle [\gamma]j \rightarrow [\gamma]j \rightarrow \emptyset</td>
<td>\langle [\gamma]j \rightarrow [\gamma]j \rightarrow \emptyset</td>
</tr>
<tr>
<td>\gamma \cdot \gamma - \gamma - 1</td>
<td>\gamma \cdot \gamma - \gamma - 1</td>
<td>\gamma \cdot \gamma - \gamma - 1</td>
<td>\gamma \cdot \gamma - \gamma - 1</td>
<td>\gamma \cdot \gamma - \gamma - 1</td>
</tr>
</tbody>
</table>

We choose the following conclusions:

- We-Well, we choose the following conclusions:
- We-Well, we choose the following conclusions:
- We-Well, we choose the following conclusions:
- We-Well, we choose the following conclusions:
- We-Well, we choose the following conclusions:

Consider the following program DINE-PHIL.

Program DINE-PHIL