Section 9: Program Algebra and Dynamic Logic
Section 9: Program Algebra and Dynamic Logic

9.1: Modelling Domain
Section 9: Program Algebra and Dynamic Logic

9.1: Modelling Domain

To be able to express statements about termination, we need one further program algebra operation.
Section 9: Program Algebra and Dynamic Logic

9.1: Modelling Domain

To be able to express statements about termination, we need one further program algebra operation.

Recall we view programs as modelled by partial functions (or more generally, binary relations).
Section 9: Program Algebra and Dynamic Logic

9.1: Modelling Domain

To be able to express statements about termination, we need one further program algebra operation.

Recall we view programs as modelled by partial functions (or more generally, binary relations).

They are only partially defined because they may not halt for some state vector inputs.
We can only express statements such as $\alpha P \beta' = 0$, which says that if an input to $P$ satisfies $\alpha$ and there is an output, then it satisfies $\beta$. 
We can only express statements such as $\alpha P \beta' = 0$, which says that if an input to $P$ satisfies $\alpha$ and there is an output, then it satisfies $\beta$.

But we’d like to be able to say $P$ does halt.
We can only express statements such as $\alpha P \beta' = 0$, which says that if an input to $P$ satisfies $\alpha$ and there is an output, then it satisfies $\beta$.

But we’d like to be able to say $P$ does halt.

For this we introduce the *domain operation* $D$. 
We can only express statements such as $\alpha P \beta' = 0$, which says that if an input to $P$ satisfies $\alpha$ and there is an output, then it satisfies $\beta$.

But we’d like to be able to say $P$ does halt.

For this we introduce the *domain operation* $D$.

This is defined for partial functions as follows:
We can only express statements such as $\alpha P \beta' = 0$, which says that if an input to $P$ satisfies $\alpha$ and there is an output, then it satisfies $\beta$.

But we’d like to be able to say $P$ does halt.

For this we introduce the \textit{domain operation} $D$.

This is defined for partial functions as follows:

$D(f)$ is the restriction of the identity function to the domain of $f$. 
We can only express statements such as $\alpha P \beta = 0$, which says that if an input to $P$ satisfies $\alpha$ and there is an output, then it satisfies $\beta$.

But we’d like to be able to say $P$ does halt.

For this we introduce the *domain operation* $D$.

This is defined for partial functions as follows:

$D(f)$ is the restriction of the identity function to the domain of $f$.

So for all $x \in X$, $D(f)(x) = x$ if $f(x)$ exists and is undefined otherwise.
So we can think of $D(f)$ as the test that asserts “$f$ halts”.
So we can think of $D(f)$ as the test that asserts “$f$ halts”.

Of course this is not generally computable, but it is algebraically useful!
So we can think of $D(f)$ as the test that asserts “$f$ halts”.

Of course this is not generally computable, but it is algebraically useful!

With the help of $D$, we can express weakest preconditions to assure a given triple is partially or totally correct.
Thus given program \( P \) and postcondition \( \beta \),
Thus given program $P$ and postcondition $\beta$, 

\{ $\alpha$ \} $P$ \{ $\beta$ \} is totally correct
Thus given program $P$ and postcondition $\beta$,

$$\{ \alpha \} \ P \ \{ \beta \} \ \text{is totally correct}$$

if and only if $\alpha \subseteq D(P\beta)$. 
Thus given program $P$ and postcondition $\beta$, 

$$\{\alpha\} \ P \ \{\beta\}$$ is totally correct

if and only if $\alpha \subseteq D(P\beta)$.

The reason is that $\{\alpha\} \ P \ \{\beta\}$ is totally correct if and only if
Thus given program $P$ and postcondition $\beta$,

$$\{ \alpha \} \ P \ \{ \beta \} \text{ is totally correct}$$

if and only if $\alpha \subseteq D(P\beta)$.

The reason is that $\{ \alpha \} \ P \ \{ \beta \}$ is totally correct if and only if $\alpha$ is a subset of the domain of $P$ such that the output is in $\beta$, 
Thus given program $P$ and postcondition $\beta$,

\[
\{ \alpha \} \ P \ \{ \beta \} \quad \text{is totally correct}
\]

if and only if $\alpha \subseteq D(P\beta)$.

The reason is that \{ $\alpha$ \} $P$ \{ $\beta$ \} is totally correct if and only if $\alpha$ is a subset of the domain of $P$ such that the output is in $\beta$,

that is, $\alpha \subseteq D(P\beta)$. 
Likewise, given program $P$ and postcondition $\beta$, 
Likewise, given program $P$ and postcondition $\beta$, 

$$\{ \alpha \} \; P \; \{ \beta \}$$

is partially correct.
Likewise, given program $P$ and postcondition $\beta$, 

$$\{ \alpha \} \ P \ \{ \beta \}$$ is partially correct 

if and only if $\alpha \subseteq D(P\beta')'$. 
Likewise, given program $P$ and postcondition $\beta$, 

$$\{ \alpha \} P \{ \beta \}$$

is partially correct 

if and only if $\alpha \subseteq D(P\beta')'$. 

The reason is that $\{ \alpha \} P \{ \beta \}$ is partially correct
Likewise, given program $P$ and postcondition $\beta$, $\{ \alpha \} \ P \ \{ \beta \}$ is partially correct if and only if $\alpha \subseteq D(P\beta')'$. The reason is that $\{ \alpha \} \ P \ \{ \beta \}$ is partially correct if and only if $\alpha$ is disjoint from the places where $P$ maps outside $\beta$, 
Likewise, given program $P$ and postcondition $\beta$,

\[
\{ \alpha \} \ P \ \{ \beta \} \text{ is partially correct if and only if } \alpha \subseteq D(P\beta')'.
\]

The reason is that \{ $\alpha$ \} $P$ \{ $\beta$ \} is partially correct if and only if $\alpha$ is disjoint from the places where $P$ maps outside $\beta$,

that is, $\alpha \cap D(P\beta') = \emptyset$, or $\alpha \subseteq D(P\beta')'$.
The notation $[P]_\beta$ is often used rather than $D(P\beta')'$. 
The notation $[P]_{\beta}$ is often used rather than $D(P_{\beta}')'$. Likewise, $\langle P \rangle_{\beta}$ is often used rather than $D(P_{\beta})$. 
The notation $[P]_\beta$ is often used rather than $D(P\beta')'$.

Likewise, $\langle P \rangle_\beta$ is often used rather than $D(P\beta)$.

These notations come from modal logic, where they are frequently used.
The notation $[P]_{\beta}$ is often used rather than $D(P_{\beta'})'$.  

Likewise, $\langle P \rangle_{\beta}$ is often used rather than $D(P_{\beta})$.  

These notations come from modal logic, where they are frequently used.  

For each program $P$, $[P]$ is a modal necessity operator,
The notation \([P]_\beta\) is often used rather than \(D(P_\beta')'\).

Likewise, \(\langle P \rangle_\beta\) is often used rather than \(D(P_\beta)\).

These notations come from modal logic, where they are frequently used.

For each program \(P\), \([P]\) is a modal necessity operator,

and \(\langle P \rangle\) is a modal possibility operator.
The notation \([P]_{\beta}\) is often used rather than \(D(P_{\beta'})'\).

Likewise, \(\langle P\rangle_{\beta}\) is often used rather than \(D(P_{\beta})\).

These notations come from modal logic, where they are frequently used.

For each program \(P\), \([P]\) is a modal necessity operator,

and \(\langle P\rangle\) is a modal possibility operator.

Note that \(\langle P\rangle_{\beta} = ([P]_{\beta'})'\).
\[ [P]^{\beta} \text{ is often called the weakest liberal precondition for the pair } P, \beta. \]
$[P] \beta$ is often called the *weakest liberal precondition* for the pair $P, \beta$.

It is “liberal” because it includes cases where termination does not occur:
\([P]_\beta\) is often called the *weakest liberal precondition* for the pair \(P, \beta\).

It is “liberal” because it includes cases where termination does not occur:

it is the weakest precondition for partial correctness.
$[P]\beta$ is often called the \textit{weakest liberal precondition} for the pair $P, \beta$.

It is “liberal” because it includes cases where termination does not occur:

it is the weakest precondition for partial correctness.

Interpreted as a predicate, $[P]\beta$ says “$\beta$ is all that can be true after $P$ is executed”.
\([P] \beta\) is often called the \textit{weakest liberal precondition} for the pair \(P, \beta\).

It is “liberal” because it includes cases where termination does not occur:

it is the weakest precondition for partial correctness.

Interpreted as a predicate, \([P] \beta\) says “\(\beta\) is all that can be true after \(P\) is executed”.

This emphasises that nothing may be true because \(P\) may not halt!
\([P] \beta\) is often called the *weakest liberal precondition* for the pair \(P, \beta\).

It is “liberal” because it includes cases where termination does not occur:

it is the weakest precondition for partial correctness.

Interpreted as a predicate, \([P] \beta\) says “\(\beta\) is all that can be true after \(P\) is executed”.

This emphasises that nothing may be true because \(P\) may not halt!

Interpreting \(<P> \beta\) as a predicate, it says “\(\beta\) will be true after \(P\) is executed”: termination is assured.
The operation $D$ can be viewed as mapping into the Boolean algebra of tests.
The operation $D$ can be viewed as mapping into the Boolean algebra of tests.

But it satisfies a number of further laws.
The operation $D$ can be viewed as mapping into the Boolean algebra of tests.

But it satisfies a number of further laws.

For all programs $P, Q, R$ and test $\alpha$: 
The operation $D$ can be viewed as mapping into the Boolean algebra of tests.

But it satisfies a number of further laws.

For all programs $P, Q, R$ and test $\alpha$:

$$D(P)P = P$$  \hspace{1cm} (D1)
$$D(P \cup Q) = D(P) \cup D(Q)$$  \hspace{1cm} (D2)
$$D(D(P)) = D(P)$$  \hspace{1cm} (D3)
$$D(PD(Q)) = D(PQ)$$  \hspace{1cm} (D4)
$$D(PQ) \subseteq D(P)$$  \hspace{1cm} (D5)
$$D(\alpha) = \alpha$$  \hspace{1cm} (D6)
The operation $D$ can be viewed as mapping into the Boolean algebra of tests.

But it satisfies a number of further laws.

For all programs $P, Q, R$ and test $\alpha$:

\[
\begin{align*}
D(P)P & = P \quad \text{(D1)} \\
D(P \cup Q) & = D(P) \cup D(Q) \quad \text{(D2)} \\
D(D(P)) & = D(P) \quad \text{(D3)} \\
D(PD(Q)) & = D(PQ) \quad \text{(D4)} \\
D(PQ) & \subseteq D(P) \quad \text{(D5)} \\
D(\alpha) & = \alpha \quad \text{(D6)}
\end{align*}
\]

(D5) says that $D(PQ)$ is contained in $D(P)$, which can be expressed as the equation $D(P) = D(P) \cup D(PQ)$, or else as $D(PQ) = D(P)D(PQ)$. 
Using program algebra with $D$, a “relative completenesss” proof for Hoare Logic may be obtained.
Using program algebra with $D$, a “relative completeness” proof for Hoare Logic may be obtained.

This means HL is as complete as predicate logic: ordinary old predicate logic is the only barrier to completeness of HL.
Using program algebra with $D$, a “relative completenesss” proof for Hoare Logic may be obtained.

This means HL is as complete as predicate logic: ordinary old predicate logic is the only barrier to completeness of HL.

Unfortunately, that is an insurmountable barrier...(see Robi’s lectures).
Using program algebra with $D$, a “relative completenesss” proof for Hoare Logic may be obtained.

This means HL is as complete as predicate logic: ordinary old predicate logic is the only barrier to completeness of HL.

Unfortunately, that is an insurmountable barrier...(see Robi’s lectures).

However, if the predicates are limited in some way, restricted completeness results exist.
Section 9.2: Introducing Dynamic Logic
Section 9.2: Introducing Dynamic Logic

*Dynamic logic* is a logical system for reasoning about programs.
Section 9.2: Introducing Dynamic Logic

*Dynamic logic* is a logical system for reasoning about programs.

It is an alternative to Hoare Logic that makes use of the modal operators $[P]$ and $\langle P \rangle$ we defined earlier.
Section 9.2: Introducing Dynamic Logic

*Dynamic logic* is a logical system for reasoning about programs.

It is an alternative to Hoare Logic that makes use of the modal operators $[P]$ and $\langle P \rangle$ we defined earlier.

Some software for program verification is based on dynamic logic.
Section 9.2: Introducing Dynamic Logic

*Dynamic logic* is a logical system for reasoning about programs.

It is an alternative to Hoare Logic that makes use of the modal operators $[P]$ and $\langle P \rangle$ we defined earlier.

Some software for program verification is based on dynamic logic.

Recall that the Hoare triple $\{ \alpha \} P \{ \beta \}$ is totally correct.
Section 9.2: Introducing Dynamic Logic

Dynamic logic is a logical system for reasoning about programs.

It is an alternative to Hoare Logic that makes use of the modal operators $[P]$ and $\langle P \rangle$ we defined earlier.

Some software for program verification is based on dynamic logic.

Recall that the Hoare triple $\{ \alpha \} P \{ \beta \}$ is totally correct if and only if $\alpha \subseteq D(P\beta)$. 

Another way to say this is that $\alpha \rightarrow \langle P \rangle \beta$ is a tautology.
Another way to say this is that $\alpha \rightarrow \langle P \rangle \beta$ is a tautology.

So we can “propositionalize” Hoare triples.
Another way to say this is that $\alpha \rightarrow \langle P \rangle \beta$ is a tautology.

So we can “propositionalize” Hoare triples.

Likewise, “ $\{ \alpha \} \ P \ \{ \beta \}$ is partially correct”
Another way to say this is that $\alpha \rightarrow \langle P \rangle \beta$ is a tautology.

So we can “propositionalize” Hoare triples.

Likewise, “ $\{ \alpha \} \; P \; \{ \beta \}$ is partially correct”

can be equivalently expressed as saying
Another way to say this is that \( \alpha \rightarrow \langle P \rangle \beta \) is a tautology.

So we can “propositionalize” Hoare triples.

Likewise, “ \( \{ \alpha \} \ P \ \{ \beta \} \) is partially correct” can be equivalently expressed as saying

“\( \alpha \rightarrow [P] \beta \) is a tautology”. 
Another way to say this is that $\alpha \rightarrow \langle P \rangle \beta$ is a tautology.

So we can “propositionalize” Hoare triples.

Likewise, “$\{ \alpha \} \ P \ \{ \beta \}$ is partially correct”

can be equivalently expressed as saying

“$\alpha \rightarrow [P] \beta$ is a tautology”.

We can write down axioms for dynamic logic that let us prove tautologies within it.
Here is one form of the axioms, expressed in terms of the "possibility operators" \( \langle P \rangle \).
Here is one form of the axioms, expressed in terms of the “possibility operators” $\langle P \rangle$.

For all programs $P, Q, R$ and tests $\alpha$: 
Here is one form of the axioms, expressed in terms of the “possibility operators” $\langle P \rangle$.

For all programs $P, Q, R$ and tests $\alpha$:

\[ \neg\langle 0 \rangle \alpha \]  
\[ \langle 1 \rangle \alpha \leftrightarrow \alpha \]  
\[ \langle P \cup Q \rangle \alpha \leftrightarrow \langle P \rangle \alpha \lor \langle Q \rangle \alpha \]  
\[ \langle P; Q \rangle \alpha \leftrightarrow \langle P \rangle \langle Q \rangle \alpha \]  
\[ \langle P^* \rangle \alpha \leftrightarrow \alpha \lor \langle P \rangle \langle P^* \rangle \alpha \]  
\[ \langle P^* \rangle \alpha \rightarrow \alpha \lor \langle P^* \rangle (\neg \alpha \land \langle P \rangle \alpha) \]
Here is one form of the axioms, expressed in terms of the “possibility operators” $⟨P⟩$.

For all programs $P, Q, R$ and tests $α$:

\[ \neg ⟨0⟩α \quad \text{(T1)} \]
\[ ⟨1⟩α \leftrightarrow α \quad \text{(T2)} \]
\[ ⟨P \cup Q⟩α \leftrightarrow ⟨P⟩α \lor ⟨Q⟩α \quad \text{(T3)} \]
\[ ⟨P; Q⟩α \leftrightarrow ⟨P⟩⟨Q⟩α \quad \text{(T4)} \]
\[ ⟨P^*⟩α \leftrightarrow α \lor ⟨P⟩⟨P^*⟩α \quad \text{(T5)} \]
\[ ⟨P^*⟩α \rightarrow α \lor ⟨P^*⟩(\neg α \land ⟨P⟩α) \quad \text{(T6)} \]

These axioms are to be viewed as tautologies.
Here is one form of the axioms, expressed in terms of the “possibility operators” \( \langle P \rangle \).

For all programs \( P, Q, R \) and tests \( \alpha \):

\[
\begin{align*}
\neg \langle 0 \rangle \alpha & \quad \text{(T1)} \\
\langle 1 \rangle \alpha & \leftrightarrow \alpha \quad \text{(T2)} \\
\langle P \cup Q \rangle \alpha & \leftrightarrow \langle P \rangle \alpha \vee \langle Q \rangle \alpha \quad \text{(T3)} \\
\langle P ; Q \rangle \alpha & \leftrightarrow \langle P \rangle \langle Q \rangle \alpha \quad \text{(T4)} \\
\langle P^* \rangle \alpha & \leftrightarrow \alpha \vee \langle P \rangle \langle P^* \rangle \alpha \quad \text{(T5)} \\
\langle P^* \rangle \alpha & \rightarrow \alpha \vee \langle P^* \rangle (\neg \alpha \wedge \langle P \rangle \alpha) \quad \text{(T6)}
\end{align*}
\]

These axioms are to be viewed as tautologies.

All can be proved using our program algebra with \( D \) (except the final one which needs a new law).
For example, (T3):
For example, (T3):

\[\langle P \cup Q \rangle_\alpha = D((P \cup Q)_\alpha)\]
For example, (T3):

\[
\langle P \cup Q \rangle \alpha = D((P \cup Q)\alpha) \\
= D(P\alpha \cup Q\alpha)
\]
For example, (T3):

\[
\langle P \cup Q \rangle_\alpha = D((P \cup Q)_\alpha) \\
= D(P_\alpha \cup Q_\alpha) \\
= D(P_\alpha) \cup D(Q_\alpha)
\]
For example, (T3):

\[ \langle P \cup Q \rangle_\alpha = D(\langle P \cup Q \rangle_\alpha) \]
\[ = D(P_\alpha \cup Q_\alpha) \]
\[ = D(P_\alpha) \cup D(Q_\alpha) \]
\[ = \langle P \rangle_\alpha \cup \langle Q \rangle_\alpha. \]
For example, (T3):

\[
\langle P \cup Q \rangle_\alpha = D((P \cup Q)_\alpha) = D(P_\alpha \cup Q_\alpha) = D(P_\alpha) \cup D(Q_\alpha) = \langle P \rangle_\alpha \cup \langle Q \rangle_\alpha.
\]

To obtain all valid tautologies we require two *rules of inference* involving formulas (of which the tautologies in the above axioms provide examples).
For example, (T3):

\[
\langle P \cup Q \rangle_\alpha = D((P \cup Q)_\alpha) \\
= D(P_\alpha \cup Q_\alpha) \\
= D(P_\alpha) \cup D(Q_\alpha) \\
= \langle P \rangle_\alpha \cup \langle Q \rangle_\alpha.
\]

To obtain all valid tautologies we require two rules of inference involving formulas (of which the tautologies in the above axioms provide examples).

*Modus ponens*: from the truth of \( \psi \) and \( \psi \rightarrow \phi \), infer the truth of \( \phi \).
For example, (T3):

\[
\langle P \cup Q \rangle_\alpha = D((P \cup Q)\alpha) \\
= D(P\alpha \cup Q\alpha) \\
= D(P\alpha) \cup D(Q\alpha) \\
= \langle P \rangle_\alpha \cup \langle Q \rangle_\alpha.
\]

To obtain all valid tautologies we require two rules of inference involving formulas (of which the tautologies in the above axioms provide examples).

*Modus ponens*: from the truth of \(\psi\) and \(\psi \to \phi\), infer the truth of \(\phi\).

In program algebra, this means showing that if \(\psi = (\psi \to \phi) = 1\), then \(\phi = 1\).
Necessitation: from the truth of $\phi$, infer the truth of $[P]\phi$. 
*Necessitation*: from the truth of $\phi$, infer the truth of $[P]\phi$.

The necessitation rule is clear from program algebra too.
Necessitation: from the truth of $\phi$, infer the truth of $[P]\phi$.

The necessitation rule is clear from program algebra too.

We want to show that if $\psi = 1$ then $[P]\psi = 1$ also.
Necessitation: from the truth of φ, infer the truth of $[P]φ$.

The necessitation rule is clear from program algebra too.

We want to show that if $ψ = 1$ then $[P]ψ = 1$ also.

But if $ψ = 1$, then $[P]ψ = D(Pψ')' = D(P0)' = D(0)' = 0' = 1$. 
Necessitation: from the truth of $\phi$, infer the truth of $[P]\phi$.

The necessitation rule is clear from program algebra too.

We want to show that if $\psi = 1$ then $[P]\psi = 1$ also.

But if $\psi = 1$, then $[P]\psi = D(P\psi')' = D(P0)' = D(0)' = 0' = 1$.

With these axioms and rules in hand, together with ways of reasoning that capture the assignment rule, dynamic logic can be used to do all the things we’ve just been doing with Hoare logic.