Modal Process Logics

Globan 06, DTU, Copenhagen, DK

Luís Caires

Departamento de Informática
Universidade Nova de Lisboa
Road Map

- **Logical Specifications**
  - Operational versus Logical Specifications
  - Logics (review)
  - Program Logics
  - Modal Logics

- **Logics for Concurrency and Distribution**
  - Hennessy-Milner Logics
  - $\mu$-Calculi
  - Spatial Logics

- **Verification Techniques**
  - Model Checking
  - Proof Systems
  - Type Systems
Models, Languages, Logics

- **Programming models**
  Specify computation by means of machines
  Automata
  Process Calculi (CCS, $\pi$-calculi,...)
  Abstract and concrete machines: partial functions, JVM, .NET

- **Programming languages**
  Specify computations by means of programming abstractions
  Lambda Calculi, Process calculi, rewriting, ...
  Expressions denote ... values, functions, objects, ...

- **Specification Logics**
  Specify requirements on machines by means of properties
  of states, computations, processes
  of computational artifacts (e.g., networks, messages)
  What properties are interesting?
“Programming” with Properties

- For design / analysis, one specifies what properties the system or implementation should satisfy.
- Properties assert constraints on states, behavior, etc.
- Some properties may be not realizable:
  - May be contradictory
  - May be non computable
  - May be not expressible in the intended model
- The meaning of a specification is a property (namely, a set of models).

\[ \llbracket \text{Spec} \rrbracket = \{ P \mid P \vDash \text{Spec} \} \]

\[ P \in \text{Spec} \iff P \vDash \text{Spec} \]
Specifications

- **Target system:** nondeterministic two register machine
  - **Set of States:** $S$ is $\mathbb{N} \times \mathbb{N}$
  - **Set of Conditions:** $C$
  - **Boot state:** $s_I \in S$
  - **Control:** Set of conditional rules $R \subseteq S \times S \times C$ ($s \rightarrow s'$ if $c$)
  - **Computation Step:** $s \rightarrow s'$ if $(s \rightarrow s'$ if $c) \in R$ and $c(s)$
  - **Computation** $C$: $s_I = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow$

- **SpecA:** $\forall C \forall s_i \in C$ if $s_i = (x_i, y_i)$ then $x_i + y_i$ is even

- **Implementation I1**
  - $s_I = (0, 0)$
  - $(x, y) \rightarrow (x+1, y+1)$
  - We have $I1 \models \text{SpecA}$ (I1 satisfies SpecA)
Specifications

- **Target system: nondeterministic two register machine**
  - Set of States: $S$ is $\mathbb{N} \times \mathbb{N}$
  - Set of Conditions: $C$
  - Boot state: $s_I \in S$
  - Control: Set of conditional rules $R \subseteq S \times S \times C$ ($s \rightarrow s'$ if $c$)
  - Computation Step: $s \rightarrow s'$ if $(s \rightarrow s' \text{ if } c) \in R$ and $c(s)$
  - Computation $C$: $s_I = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow$

- **SpecA**: $\forall C \forall s_i \in C$ if $s_i = (x_i, y_i)$ then $x_i + y_i$ is even

- **Implementation I2**
  - $s_I = (1,1)$
  - $(x, y) \rightarrow (x+x, y+y)$
  - $(x, y) \rightarrow (x-1, y+1)$ if $(x>0)$

- We also have $I2 \models SpecA$
Specifications

- **Target system**: nondeterministic two register machine
  - Set of States: $S$ is $\mathbb{N} \times \mathbb{N}$
  - Set of Conditions: $C$
  - Boot state: $s_I \in S$
  - Control: Set of conditional rules $R \subseteq S \times S \times C$ ($s \rightarrow s'$ if $c$)
  - Computation Step: $s \rightarrow s'$ if ($s \rightarrow s'$ if $c$) $\in R$ and $c(s)$
  - Computation $C$: $s_I = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow$

- **SpecA**: $\forall C \forall s_i \in C$ if $s_i = (x_i, y_i)$ then $x_i + y_i$ is even
- **SpecB**: $\exists C \exists s_k \in C$ if $s_k = (x_k, y_k)$ then $y_k > x_k$

- **Implementation I2**
  - $s_I = (1,1)$
  - $(x,y) \rightarrow (x+x, y+y)$
  - $(x,y) \rightarrow (x-1, y+1)$ (if $x > 0$)

- So $I_2 \models SpecA$; $I_2 \models SpecB$; but not $I_1 \models SpecB$
“Programming” with Properties

**SpecA:** \( \forall C \ \forall s_i \in C \text{ if } s_i = (x_i, y_i) \text{ then } x_i + y_i \text{ is even} \)

**SpecB:** \( \exists C \ \exists s_k \in C \text{ if } s_k = (x_k, y_k) \text{ then } y_k > x_k \)

**Implementation I1**
- \( s_i = (0, 0) \)
- \( (x, y) \rightarrow (x+1, y+1) \)

**Implementation I2**
- \( s_i = (1, 1) \)
  - \( (x, y) \rightarrow (x+x, y+y) \)
  - \( (x, y) \rightarrow (x-1, y+1) \) (if \( x > 0 \))

**SpecA \land SpecB** refines **SpecA**

**SpecC:** \( \exists C \ \exists s_k \in C \ s_k = (x_k, y_k) \text{ & } y_k > x_k \text{ & } x_k + y_k \text{ is even} \)

**SpecA \land SpecB** entails **SpecC**
Operational specifications are naturally monolithic
An operational specification specifies just one model
Always realizable by definition
Good to guide software construction
Good for analysis

A logic is a language to express properties (a.k.a. sets of systems / programs)
Logical specifications are naturally modular
A logical specification specifies a class of models
May be not realizable
May be specialized (by refinement)
Good for design, verification, and analysis
What logics are there ...?
From Logic to Modal Logic
Propositional Logic

Syntax
A set $\mathcal{A}$ of atomic *propositions* (basic properties)

$$\mathcal{A}, \mathcal{B}, \mathcal{C} ::= a \in \mathcal{A} | \mathcal{A} \land \mathcal{B} | \neg \mathcal{A} | \text{True}$$

Semantics
The *universe*: a nonempty set $\mathcal{U}$ of *individuals*

A *valuation*: $\nu: \mathcal{A} \to \wp(\mathcal{U})$

A formula $\mathcal{A}$ *expresses* a property $[\mathcal{A}]_\nu \subseteq \mathcal{U}$

$$[a]_\nu = \nu(a)$$

$$[\mathcal{A} \land \mathcal{B}]_\nu = [\mathcal{A}]_\nu \cap [\mathcal{B}]_\nu$$

$$[\text{True}]_\nu = \mathcal{U}$$

$$[\neg \mathcal{A}]_\nu = \mathcal{U} \setminus [\mathcal{A}]_\nu$$
Propositional Logic

Syntax

A set $A$ of atomic propositions (basic properties)

$$A, B, C ::= \text{a} \in A \mid A \land B \mid \neg A \mid \text{True}$$

Abbreviations ...

False $\triangleq \neg\text{True}$

$$A \lor B \triangleq \neg(\neg A \land \neg B)$$

$$A \Rightarrow B \triangleq \neg A \lor B$$

Assertions (judgements; sequents)

$$A_1, \ldots, A_n \vdash B_1, \ldots, B_m$$

Validity

$$\text{valid } (A \vdash B) \triangleq \forall u.\forall v u. \quad \left[ \land_i A_i \right]_v \subseteq \left[ \lor_i B_i \right]_v$$

$$\forall u.\forall v u. \quad \left[ \neg A \lor B \right]_v = u$$
Propositional Logic

A Proof System (Sequent Calculus)

\[ \begin{align*}
A & \vdash A \\
A, C & \vdash B & A & \vdash C, B & C & \vdash D, A & C & \vdash D, A \\
A & \vdash \neg C, B & A & , \neg C & \vdash B & C & \vdash D, A \land B & C, A, B & \vdash D \\
C & \vdash D & C & \vdash D & C, A, B, \vdash D & C, A \land B & \vdash D \\
C, A & \vdash D & C, A & \vdash D, A \\
C, A, A & \vdash D & C & \vdash D, A & C & \vdash D, A, A \\
\end{align*} \]

Soundness: if \( A \vdash B \) then valid (\( A \vdash B \))

Completeness: if valid (\( A \vdash B \)) then \( A \vdash B \)

Decidability: we can decide \( A \vdash B \)
Predicate Logic

**Syntax**
- A set $V$ of variables ($x, y, z$)
- A set $P$ of atomic predicates (p,q,r) (basic relations)

$$A, B, C ::= p(x,y) \mid A \land B \mid \neg A \mid \forall x.A \mid \text{True}$$

**Semantics**
- The universe: a nonempty set $U$ of individuals
- A interpretation: $I: P \rightarrow \wp(U \times U)$
- A valuation: $v: V \rightarrow U$

**Satisfaction** (a model $M$ satisfies a formula $A$)

$$M \models A$$

A formula denotes (specifies) a set of models

$$[A] = \{ I;v \mid I;v \models A \}$$
**Predicate Logic**

### Semantics

The universe: a nonempty set $\mathcal{U}$ of individuals $(a, b)$

A interpretation: $I: P \rightarrow \mathcal{P}(\mathcal{U} \times \mathcal{U})$

A valuation: $v: V \rightarrow \mathcal{U}$

A model: $M = (I; v)$

### Satisfaction

$I; v \models p(x, y)$ iff $(v(x), v(y)) \in I(p)$

$M \models \neg A$ iff not $M \models A$

$M \models A \land B$ iff $M \models A$ and $M \models B$

$I; v \models \forall x. A$ iff $\forall a \in \mathcal{U}. I; v\{x/a\} \models A$

### Valid

$\text{valid } (A_1, ..., A_n \vdash B_1, ..., B_m) \triangleq [\land_i A_i] \subseteq [\lor_i B_i]$
Predicate Logic

A Proof System

\[
\begin{align*}
(x \text{ not free in } C, D) & \quad C \vdash D, \forall x.\neg \gamma \\
C \vdash D, \forall x.\neg \gamma & \quad C \vdash D, \forall x.\neg \gamma \\
C \vdash D, \forall x.\neg \gamma & \quad C, \forall x.\neg \gamma \vdash D
\end{align*}
\]

Soundness: if \( \mathcal{A} \vdash \mathcal{B} \) then valid ( \( \mathcal{A} \vdash \mathcal{B} \) )

Completeness: if valid ( \( \mathcal{A} \vdash \mathcal{B} \) ) then \( \mathcal{A} \vdash \mathcal{B} \)

Undecidability: no algorithm can decide \( \mathcal{A} \vdash \mathcal{B} \)

One can use predicate logic to reason about quite a lot, and certainly about programs, processes, networks,...

But we aim at something more specialized.

The computer science approach: seek and analyze the “right” (preferably tractable) abstractions.
Hoare Logic

Simple imperative programs

\[ P ::= x := E \mid \text{if } E \text{ then } P \text{ else } P \mid \text{while } E \text{ do } P \]

Assertions (Hoare triples)

\[ \{ \mathcal{A} \} \ P \ \{ \mathcal{B} \} \]

\( \mathcal{A}, \mathcal{B} \) are state formulas (predicate logic formulas)

\( \mathcal{A} \) is the precondition, assumed to hold before \( P \) runs

\( \mathcal{B} \) is the postcondition, concluded to hold after \( P \) terminates

Examples

\[ \{ x = 0 \} \ x := x+1 \ \{ x > 0 \} \]

\[ \{ n \geq 0 \} \ x := 1; \ i := 0; \text{ while } i < n \text{ do } (i = i+1; \ x := x \cdot i) \ \{ x = n! \} \]

N.B. Names in formulas \((x, i, n)\) refer to program variables
**Hoare Logic**

- A set of program variables $\mathcal{V}$
- A state is a valuation $s : \mathcal{V} \rightarrow \text{int}$

**Semantics of programs**

**Expression evaluation:** $[\cdot]_s$ maps $E$ to $[E]_s$

$$
[n]_s = n ; [x]_s = s(x); [E + F]_s = [E]_s + [F]_s; \text{ etc ...}
$$

**Transition relation:** go from $s$ to $r$ by running $P$: $s \xrightarrow{P} r$

$$
\begin{align*}
\text{x:=E} & \quad s \xrightarrow{x / [E]_s} s \{x / [E]_s\} \\
[E]_s = \text{true} & \quad s \xrightarrow{P} r \quad r \xrightarrow{E \text{ do } P} p \\
\text{while E do } P & \quad s \xrightarrow{P} r \\
\text{while E do } P & \quad r \xrightarrow{Q} p \\
\text{while E do } P & \quad s \xrightarrow{P;Q} p
\end{align*}
$$
Hoare Logic

Semantics of assertions

\[ \text{valid} \left( \{ A \} \ P \ \{ B \} \right) \triangleq \]
\[ \forall s. \text{ if } s \models A \text{ and } s \xrightarrow{P} r \text{ then } r \models B \]

Proof System

Some proof rules such as

\[
\begin{align*}
\{ A \land E \} & \ P \ \{ A \} \\
\{ A \} & \text{ while } E \text{ do } P \ \{ A \land \neg E \} \\
\end{align*}
\]

\[
\begin{align*}
A \ A' & \ \{ A' \} \ P \ \{ B' \} \ B' \ B \\
\{ A \} & \ P \ \{ B \} \\
\end{align*}
\]

Useful to write safety specifications

Do not ensure the program will actually do something:
If \( P \) does not terminate, then \( \{ A \} \ P \ \{ B \} \) is valid for any \( A, B \)
But if something happens, all will be ok (if the spec is)
Modal Logic as Program Logic

- Modal logics talk about structures consisting of many suitably related states (or “worlds”).
  Each world is a “classical” model (e.g., a boolean algebra).

  e.g., $s \models \mathcal{A}$

  The novelty: special operators (called modalities) allowing us to “jump” from world to world, or quantify over worlds.

  e.g., and $s \rightarrow r$ and $r \models \mathcal{A}$ then $s \models \lbrack\text{next}\rbrack\mathcal{A}$

- Specific modal logics may talk about time, behavior, space, resources, data, knowledge, necessity, etc...

- “Program Logic” as a modal logic:

  $\mathcal{A}, \mathcal{B}, \mathcal{C} ::= p(x,y) \mid \mathcal{A} \land \mathcal{B} \mid \neg \mathcal{A} \mid \forall x.\mathcal{A} \mid \lbrack P \rbrack\mathcal{A}$

  $s \models \lbrack P \rbrack\mathcal{A}$ iff $\forall r.\ if\ s \xrightarrow{P} r\ then\ r \models \mathcal{A}$
Modal Logic (Classical)

- **Worlds**
  Intuition: a “world” is a state \( s \) (a boolean valuation)

- **Accessibility** (relation between worlds)
  A transition relation \( s \rightarrow r \)
  Intuition: for each world \( s \), there are some “alternative” worlds, namely those worlds \( r \) such that \( s \rightarrow r \)

- **Syntax**
  \[ \mathcal{A}, \mathcal{B}, \mathcal{C} ::= a \in \mathcal{A} | \mathcal{A} \land \mathcal{B} | \neg \mathcal{A} | \text{True} | \Box \mathcal{A} \]

- **Models** \( \mathcal{M} = (v, \rightarrow) \)
  A valuation \( v: \mathcal{A} \rightarrow \wp(S) \)
  Says what holds at each world
  A transition system \( \rightarrow \)
  Says what worlds are compatible / nearby / next / ...
Modal Logic (Classical)

- **Model** $\mathcal{M}$
  
  A valuation $\nu: A \rightarrow \wp(S)$
  
  A transition system $\rightarrow$

- **Semantics**
  
  $s \models a$ iff $s \in \nu(a)$
  
  $s \models \Box A$ iff $\forall r. \text{ if } s \rightarrow r \text{ then } r \models A$

  **valid** $(A_1, \ldots, A_n \vdash B_1, \ldots, B_m) \triangleq [\land_i A_i] \subseteq [\lor_i B_i]$

- **Examples of (abstract) modal reasoning**
  
  **valid** $(A)$ implies valid $(\Box A)$ (Necessitation)

  $\Box (A \Rightarrow B) \Rightarrow (\Box A \Rightarrow \Box B)$ (Axiom K)

  $\Box A \Rightarrow A$ (Axiom T; is this sensible?)

- **What are the axioms? (please specify your model ...)**
More Modal Logics

- **Linear Time Temporal Logic**
  - Models are a time line
  - Amir Pnueli proposed LTL (1977) for reasoning about concurrent and non terminating programs such as operating systems.

- **Branching Time Temporal Logic**
  - Models are trees, each instant may have different futures
  - Useful to reason about non-determinism

- **Computational Tree Logic**
  - Models are trees
  - CTL distinguishes between “path” and “state” modalities

- **Process Logics**
  - Hennessy-Milner Logics
  - μ-Calculus
  - Spatial Logics
Hennessy-Milner Logics
A modal logic for labeled transition systems

Labeled transition system

- A set $A$ of actions $(\alpha, \beta)$
- A set $S$ of states
- A labeled transition relation $T \subseteq S \times A \times S$

N.B. Write $s \xrightarrow{\alpha} r$ when $(s, \alpha, r) \in T$

Syntax

$$\mathcal{A}, \mathcal{B}, \mathcal{C} ::= \bigwedge_{i \in I} \mathcal{A}_i \mid \neg \mathcal{A} \mid \langle \alpha \rangle \mathcal{A}$$

Remarks

Infinitary syntax: True $\triangleq \bigwedge_{i \in \emptyset} \mathcal{A}_i$

No propositional symbols; the logic just observes actions

Hint: **HML** is a modal logic of pure behavior
Hennessy-Milner Logic

A modal logic for labeled transition systems

Labeled transition system
- A set $A$ of actions $(\alpha, \beta)$
- A set $S$ of states
- A labeled transition relation $T \subseteq S \times A \times S$
- N.B. Write $s \xrightarrow{\alpha} r$ when $(s, \alpha, r) \in T$

Syntax (finitary version)

$$A, B, C ::= A \land B \mid \neg A \mid \langle \alpha \rangle A$$

Modalities $\langle \alpha \rangle A$ observe nearby states

In some $\alpha$-next state $A$ holds: $\langle \alpha \rangle A$

In some $\alpha$-next states $A$ holds: $[\alpha] A \models \neg \langle \alpha \rangle \neg A$
Hennessy-Milner Logic

A modal logic for labeled transition systems

Labeled transition system
A set $A$ of actions $(\alpha, \beta)$
A set $S$ of states
A labeled transition relation $T \subseteq S \times A \times S$
N.B. Write $s \xrightarrow{\alpha} r$ when $(s, \alpha, r) \in T$

Satisfaction

\begin{align*}
  s \models (\alpha)A & \iff \exists r. \ s \xrightarrow{\alpha} r \text{ and } r \models A \\
  s \models \neg A & \iff \neg s \models A \\
  s \models A \land B & \iff s \models A \text{ and } s \models B
\end{align*}
Hennessy-Milner Logic (CCS)

- Calculus of Communicating Systems (Milner)
  A set $A$ of actions $(\alpha, \tau)$ where $\alpha = n$ or $\alpha = \overline{n}$
  A set $P$ of processes

$$
P, Q, R ::= 0 \mid P \mid Q \mid \alpha.P \mid (\text{new } n)P \mid x \mid \text{rec } x.P
$$

- Labeled Transition System for CCS

$$
\begin{align*}
\alpha.P & \xrightarrow{\alpha} P \\
\alpha & \text{ in } \alpha \\
(\text{n not in } \alpha) & \\
\alpha & \text{ in } \alpha \\
\alpha & \text{ in } \alpha
\end{align*}
$$

- We interpret Hennessy-Milner Logic on CCS
Hennessy-Milner Logic (CCS)

- \( P \models \langle n \rangle \text{True} \)
  - \( P \) can perform an output on \( n \)

- \( P \models [n] \text{False} \)
  - \( P \) refuses to perform an input on \( n \)

- \( P \models \langle n \rangle [n] \text{False} \)
  - \( P \) can input on \( n \), and then refuse to perform an input on \( n \)

- \( P \models \langle n \rangle \text{True} \land [n][\tau] \text{False} \)
  - \( P \) can input on \( n \), but after any such input will get stuck

- \( P \models \langle n \rangle \langle m \rangle \text{True} \land \langle n \rangle [m] \text{False} \)
  - \( P \) can do \( n \) and then do \( m \), but also do \( n \) and after refuse \( m \)

- \( P \models [n] \langle m \rangle \text{True} \land [n][m] \text{False} \)
  - \( P \) can do \( m \) after any \( n \), but also refuse \( m \) after any \( n \)
  - So, \( P \) cannot really do \( n \)

So, \( P \) cannot really do \( n \)
Hennessy-Milner Logic

Other useful modalities
All definable in the infinitary logic (how?)

\[ \mathcal{A}, \mathcal{B}, \mathcal{C} ::= \mathcal{A} \land \mathcal{B} \mid \neg \mathcal{A} \mid (\mathcal{S})\mathcal{A} \mid (\ast)\mathcal{A} \mid (\neg \mathcal{S})\mathcal{A} \]

Satisfaction

\[ s \models (\mathcal{S})\mathcal{A} \iff \text{exists } r, \alpha \in S \text{ and } s \xrightarrow{\alpha} r \text{ and } r \models \mathcal{A} \]

\[ s \models (\ast)\mathcal{A} \iff \text{exists } r, \beta. \; s \xrightarrow{\beta} r \text{ and } r \models \mathcal{A} \]

\[ s \models (\neg \mathcal{S})\mathcal{A} \iff \text{exists } r, \alpha \notin S \text{ and } s \xrightarrow{\alpha} r \text{ and } r \models \mathcal{A} \]

\[ s \models \neg \mathcal{A} \iff \text{not } s \models \mathcal{A} \]

\[ s \models \mathcal{A} \land \mathcal{B} \iff s \models \mathcal{A} \text{ and } s \models \mathcal{B} \]
Find a set of HML formulas that precisely characterize the labeled transition system $T$ below:

The specification thus obtained is usually called a “characteristic formula” of $T$. 
What can we say about?

Express in HML some properties of the processes

\[ P_1 \triangleq \overline{n}.(n.0 \mid \overline{n}.0) \]

\[ P_2 \triangleq \text{rec } x. n.(\overline{n}.0 \mid x) \]

\[ P_3 \triangleq (\text{new } n)(P_1 \mid P_2) \]
Can we find a model of?

Try to find CCS models for the HML formulas below

\[ \mathcal{F} \triangleq \langle n \rangle \langle m \rangle \text{True} \land \langle n \rangle [ m ] \text{False} \]

\[ \mathcal{G} \triangleq \langle n \rangle \langle \overline{n} \rangle \text{True} \land [ \tau ] [ * ] \text{False} \]

\[ \mathcal{H} \triangleq [ -\{ \overline{n}, \tau \} ] \text{False} \land \langle \tau \rangle \langle n \rangle \text{True} \]
Can we distinguish by HML properties?

\[ T_1 \cong \]

\[ T_2 \cong \]
Can we distinguish by HML properties?

\[ T3 \trianglelefteq \]

\[ T4 \trianglerighteq \]
Indistinguishability in a general modal logic $L$

States $s$ and $r$ are indistinguishable in $L$ if
\[ \forall \mathcal{A}. s \models \mathcal{A} \text{ iff } r \models \mathcal{A}. \]

We define logical equivalence of states, noted $=_{L}$, by
\[ s =_{L} r \iff \forall \mathcal{A}. s \models \mathcal{A} \text{ iff } r \models \mathcal{A}. \]

Logical equivalence is an equivalence relation on $S$

A logic $L$ is finer (has more separation power) than a logic $L'$ if $=_{L} \subseteq =_{L'}$

Given a property $\mathcal{P} \subseteq S$, we say $\mathcal{P}$ is expressible in $L$ if there is a formula $\mathcal{A}$ of $L$ such that $[\mathcal{A}] = \mathcal{P}$

A logic $L$ is more expressive than a logic $L'$ if every property expressible in $L'$ may be also expressed in $L$
Bisimulation

- Caracterizes coinductively indistinguishable states. A binary relation $B \subseteq S \times S$ is a bisimulation if for all $(s, r) \in B$:
  - if $s \xrightarrow{\alpha} s'$ then there is $r'$ such that $r \xrightarrow{\alpha} r'$ and $(s', r') \in B$
  - if $r \xrightarrow{\alpha} r'$ then there is $s'$ such that $s \xrightarrow{\alpha} s'$ and $(s', r') \in B$

- Bisimulations are equivalence relations.
- Bisimulations are closed under arbitrary unions.
- Bisimilarity $\sim$ is the greatest bisimulation.
  \[ \sim = \bigcup \{ B \mid B \text{ is a bisimulation} \} \]

- We may define similar notions for most modal models (Kripke models); e.g., we may also want to observe state valuations, etc, ...
**Separation Power of HML**

**Theorem**

If $P \sim Q$ then $P =_L Q$

This follows from

**Lemma**

For any formula $\mathcal{A}$, if $P \models \mathcal{A}$ and $P \sim Q$ then $Q \models \mathcal{A}$

Proof: induction on the structure of $\mathcal{A}$.

HML does not distinguish between bisimilar states

N.B. Analogous results should hold for most modal logics, given a suitable notion of bisimulation.

HML is extensional with relation to pure behaviors.

Some interesting properties of processes are not extensional (deadlock, stuckness, race freeness).
Separation Power of HML

Theorem

If $P =_L Q$ then $P \sim Q$

This follows from

Lemma

$=_L$ is a strong bisimulation

Proof: it may be more easy to show the converse:

If $P \sim Q$ then $P \neq_L Q$

HML can only observe processes up to a finite depth

If $P \models \mathcal{A}$ and $P \sim_k Q$ then $Q \models \mathcal{A}$ for $k \geq \text{size}(\mathcal{A})$

So, we may have $P \sim_k Q$ and $P \sim Q$.

N.B. If $P \sim Q$ then there is $k$ such that $P \sim_k Q$. 
Specifying an infinite amount of information

\[ L \triangleq \langle \text{dead} \rangle \text{dead} \land \langle \text{b} \rangle \text{True} \]

\[ \text{dead} \triangleq [\ast] \text{False} \]

\[ \{ \text{a} \} \text{dead} \land \{ \text{b} \} \text{True} \]

\[ \{ \text{a} \} \text{dead} \land \{ \text{b} \}(\{ \text{a} \} \text{dead} \land \{ \text{b} \} \text{True}) \]

\[ \{ \text{a} \} \text{dead} \land \{ \text{b} \}(\{ \text{a} \} \text{dead} \land \{ \text{b} \}(\{ \text{a} \} \text{dead} \land \{ \text{b} \} \ldots )) \]

\[ \nu X. (\{ \text{a} \} \text{dead} \land \{ \text{b} \} X) \]
The $\mu$-Calculus (Kozen)

**Syntax (extension of HML with fixed points operator)**

A set $\mathcal{V}$ of propositional variables $(x, y, z)$

$\mathcal{A}, \mathcal{B}, \mathcal{C} ::= \mathcal{A} \land \mathcal{B} \mid \neg \mathcal{A} \mid \langle \alpha \rangle \mathcal{A} \mid \nu X. \mathcal{A} \mid X \ (\in \mathcal{V})$

**Satisfaction**

A valuation: $\nu: \mathcal{V} \rightarrow \wp(S)$

$s \models_{\nu} \langle \alpha \rangle \mathcal{A}$ iff exists $r. \ s \xrightarrow{\alpha} r$ and $r \models_{\nu} \mathcal{A}$

$s \models_{\nu} \neg \mathcal{A}$ iff not $s \models_{\nu} \mathcal{A}$

$s \models_{\nu} \mathcal{A} \land \mathcal{B}$ iff $s \models_{\nu} \mathcal{A}$ and $s \models_{\nu} \mathcal{B}$

$s \models_{\nu} X$ iff $s \in \nu(X)$

$s \models_{\nu} \nu X. \mathcal{A}$ iff $s \in \text{gfp}(\lambda \mathcal{P}. [\mathcal{A}]_{\nu}[X/\mathcal{P}])$

$[\nu X. \mathcal{A}]_{\nu} = \cup \{\mathcal{P} \subseteq S \mid \mathcal{P} \subseteq [\mathcal{A}]_{\nu}[X/\mathcal{P}]\}$
The $\mu$-Calculus

- **Least fixed point**
  \[
  \mu X. A \triangleq \neg \nu X. \neg A \{X/\neg X\}
  \]

- **Always $A$ (under the actions in $S$)**
  \[
  \text{inv } A \triangleq \nu X. ( A \land [S]X )
  \]
  useful to specify invariant properties of systems

- **Possibly $A$ (after some actions in $S$)**
  \[
  \text{poss } A \triangleq \mu X. ( A \land (S)X )
  \]
  N.B. $\text{poss } A = \neg \text{inv } \neg A$

- **$A$ until $B$ (under the actions in $S$)**
  \[
  A \text{ until } B \triangleq \nu X. ( B \lor ( A \land [S]X ) )
  \]
The µ-Calculus

- Eventually \( A \) (after some actions in \( S \))
  \[
  \text{ev } A \triangleq \mu X. (A \land \langle S \rangle \text{True } \land [S]X)
  \]

- \( A \) until \( B \) (under the actions in \( S \))
  \[
  A \text{ until } B \triangleq \nu X. (B \lor (A \land \langle S \rangle \text{True } \land [S]X))
  \]

- A process is **insistent** for action \( s \) if in every infinite computation sequence, \( s \) is executed infinitely often.
  \[
  \text{insistent } s \triangleq
  \]

- A process is **unfair** w.r.t. the action \( s \) if it may always perform \( s \), but in some possible infinite computation sequence it never actually gets to perform \( s \).
  \[
  \text{unfair } s \triangleq
  \]
Spatial Logics for Concurrency
Reasoning about Distributed Systems

- **Traditional focus**
  - Abstract from irrelevant implementation details
  - Study extensional models of *processes as pure behaviors*

- **A focus on Distributed Systems**
  - Systems where behavior is *spatially* distributed among sites
  - Processes behave in time, but site and move in space
  - Structure of space may change during computation
  - Non-behavioral aspects just cannot be abstracted way
    - *E.g.*, geometry, topology, identity, naming, …
  - Several kinds of spatial structure …
  - Several possibilities for space / behaviour interaction …

- **Operational Techniques**
  - Spatial properties also useful for compositional reasoning
  - Spatial logics can also provide a basis for type systems
Example: Resource Discovery

A Distributed Directory Protocol
A Distributed Directory Protocol
Example: Resource Discovery

A Distributed Directory Protocol
“The connection structure is a spanning tree”
N.B.: Key for proving correctness of the protocol.
“Some link failed; the network is partitioned”
“Some site crashed; the network is partitioned”
"A new peer joined in"
“It is always possible for any site to eventually acquire exclusive access to the resource”
Concurrency (spatial) monoid $\langle \text{Procs}, 0, \mid \rangle$

Spatial identity is (close to) structural congruence

Name Restriction $(\forall n)P$
Directory $\triangleq (\forall \text{o}_b) (A | B | C | D | E | F)$
Names and Spatial Structure

- Names identify **resources** in a (spatial) scope.
- Uses of names may be either **public** or **hidden**:
  \[ n \in fn(P) \text{ if and only if } \neg \exists Q. P \equiv (\forall n)Q \]
- A name always splits a system in two parts
- Hidden names can induce spatial bonds:
**Names and Spatial Structure**

- *Pure names*, as construed by Roger Needham [N89], abstract general purpose atomic data.

- Names name resources, *e.g.*, values, communication channels, secret keys, nonces, in a (spatial) **scope**.

- Uses of names may be either **public** or **hidden**:

  \[ n \in fn(P) \text{ if and only if } \neg \exists Q. P \equiv (\forall n)Q \]

- Hidden names may induce spatial bonds:
Concurrency (spatial) monoid $\langle \text{Procs}, 0, \mid \rangle$

Tree structure (cf., Mobile Ambients): $n[P]$

Name Restriction: $(\nu n)P$

Bigraphical Structure (Milner)
Properties of Distributed Models

- **Temporal & Hennessy-Milner Logics**
  - Modal logics with modalities for observing temporal structure
  - Useful for specifying general safety and liveness properties
  - Do not distinguish between bisimilar processes

- **Spatial Logics [CM98,CG00,CC02-04,C04]**
  - Modal logics with modalities for observing spatial structure
  - Each “world” is a structured space
  - “Space” is seen as a kind of resource (logics can separate, count)

- **Spatial Observations**
  - Not invariant under traditional behavioral equivalences
    
\[
 n.0 + m.0 \not\approx_L n.0 \mid m.0
\]
  - Intensionality? (more later)
  - Invariant under a natural notion of spatial equivalence
    
\[
 \approx \text{structural congruence} [\text{San01}]
\]
    
\[
 \approx \text{extended structural congruence} [\text{Cai04}]
\]
The π-Calculus

\[ n, m, p \in \text{Names} \]

\[ P, Q \in \text{Procs} ::= \text{Processes} \]

\[
\begin{align*}
0 & \quad \text{Void} \\
P \mid Q & \quad \text{Composition} \\
(\nu n)P & \quad \text{Restriction} \\
n!(m).P & \quad \text{Output} \\
n?(m).P & \quad \text{Input} \\
\Sigma \alpha_i.P_i & \quad \text{Choice} \\
(rec \ X[x].P)[m] & \quad \text{Recursion} \\
X[m] & \quad \text{Variable}
\end{align*}
\]

**Reduction ( \( P \rightarrow Q \) )**:

\[
\begin{align*}
m!(n).P \mid m?(p).Q & \rightarrow P \mid Q[p/n] \\
P \rightarrow Q & \text{ implies } (\nu n)P \rightarrow (\nu n)Q \\
P \rightarrow Q & \text{ implies } P \mid R \rightarrow Q \mid R \\
P \equiv P', P' \rightarrow Q', Q' \equiv Q & \text{ implies } P \rightarrow Q
\end{align*}
\]

**Spatial Congruence**: 

\[
\begin{align*}
P \mid 0 & \equiv P \\
P \mid Q & \equiv Q \mid P \\
(P \mid Q) \mid R & \equiv P \mid (Q \mid R) \\
(\nu n)0 & \equiv 0 \\
(\nu n)(\nu m)P & \equiv (\nu m)(\nu n)P \\
(\nu n)(P \mid Q) & \equiv P \mid (\nu n)Q \\
& \quad \text{if } n \notin \text{fn}(P) \\
(rec \ X[x].P)[m] & \equiv \\
P \{x / m\} \{X / (rec \ X[x].P)}
\end{align*}
\]
The $\pi$-Calculus

$n,m,p \in \text{Names}$

$P,Q \in \text{Procs} ::= \text{Processes}$

- $0$\hspace{1cm}Void
- $P \mid Q$\hspace{1cm}Composition
- $(\forall n)P$\hspace{1cm}Restriction
- $n!(m).P$\hspace{1cm}Output
- $n?(m).P$\hspace{1cm}Input
- $\sum \alpha_i.P_i$\hspace{1cm}Choice
- $(\text{rec } X[x].P)[m]$\hspace{1cm}Recursion
- $X[m]$\hspace{1cm}Variable

**Interaction** $(P \rightarrow Q)$:

- $m, n \notin p$
  - $n!(m)$\hspace{1cm}$(\forall p)(n!(m).Q \mid P) \rightarrow (\forall p)(Q \mid P)$
- $m, n \notin p$
  - $n?(m)$\hspace{1cm}$(\forall p)(Q \mid n?(q).P) \rightarrow (\forall p)(Q|P\{q\leftarrow m\})$

**Spatial Congruence:**

- $P \mid 0 \equiv P$
- $P \mid Q \equiv Q \mid P$
- $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
- $(\forall n)0 \equiv 0$
- $(\forall n)(\forall m) P \equiv (\forall m)(\forall n) P$
- $(\forall n)(P \mid Q) \equiv P \mid (\forall n)Q$
  - if $n \notin \text{fn}(P)$
- $(\text{rec } X[x].P)[m] \equiv$
- $P \{x / m\} \{X / (\text{rec } X[x].P)\}$
Processes behave by communicating:

\[ n!(\text{msg}).Q \mid n?(x).P \xrightarrow{\tau} Q \mid P\{ x/\text{msg} \} \]

Internal

\[ P \xrightarrow{\tau} Q \]

Output

\[ P \xrightarrow{n!(m)} Q \]

Input

\[ P \xrightarrow{n?(m)} Q \]

\[ \alpha ::= n!(m) \mid n?(m) \mid \tau \]

Hennessy-Milner like modalities:

\[ \langle \alpha \rangle \mathcal{A} \]

\[ P \models \langle \alpha \rangle \mathcal{A} \text{ iff } P \xrightarrow{\alpha} Q \text{ and } Q \models \mathcal{A} \]
Spatial Observations

- Composition and restriction are interpreted as spatial rather than dynamic operations. E.g.,

- Any process $P$ can be decomposed in several ways into a pair $(Q, R)$ such that the spatial identity holds:

$$P \equiv Q | R$$

$P \rightarrow (Q, R)$

$\equiv \text{“spatial congruence”}$

$$P | 0 \equiv P$$

$$P | Q \equiv Q | P$$

$$P | Q) | R \equiv P | (Q | R)$$
Composition and restriction are interpreted as spatial rather than dynamic operations. E.g.,

Any process $P$ can be decomposed in several ways into a pair $(n, Q)$ such that the spatial identity holds:

$$P \equiv (\forall n)Q$$

≡ “spatial congruence”

$$(\forall n)0 \equiv 0$$

$$(\forall n)(\forall m)P \equiv (\forall m)(\forall n)P$$

$$(\forall n)(P | Q) \equiv P | (\forall n)Q$$
\[\pi\text{-Calculus System Observations}\]

- **Composition**
  \[P \parallel Q, R\]

- **Restriction**
  \[P \Rightarrow n, Q\]

- **Internal Action**
  \[P \xrightarrow{\tau} Q\]

- **Input Action**
  \[P \xrightarrow{n?m} Q\]

- **Output Action**
  \[P \xrightarrow{n!m} Q\]

\[n, m \in \text{Names}\]
\[\alpha \in \text{Actions} \; ::= \; \tau \; \text{Internal}\]
\[n?(m) \; \text{Input Action}\]
\[n!(m) \; \text{Output Action}\]
A core Spatial-Behavioral Logic

\[ A \land B, \neg A, \ldots \quad \text{Boolean Operators} \quad P \models \mathcal{A} \]

0 \quad \text{Void}

\[ A | B \quad \text{Composition} \]

\[ H_x.A \quad \text{Hidden Name Quantifier} \]

\[ @n \quad \text{Free Name Occurrence} \]

\[ m = n \quad \text{Name Equality} \]

\[ \langle \alpha \rangle A \quad \text{Action} \quad (\alpha \in \text{Actions}) \]

\[ \forall x.A \quad \text{Universal Quantifier} \]

\[ \nu X.A \quad \text{Recursion} \quad \text{(Greatest Fixed Point)} \]
\[
P \models 0 \quad \text{iff} \quad P \equiv 0
\]

\[
P \models \mathcal{A} \mid \mathcal{B} \quad \text{iff} \quad \exists Q, R, P \equiv Q \mid R \text{ and } Q \models \mathcal{A} \text{ and } R \models \mathcal{B}
\]

\[
P \models \mathsf{H}_x.\mathcal{A} \quad \text{iff} \quad \exists Q, n \not\equiv \mathcal{A}, P \equiv (\forall n)Q \text{ and } Q \models \mathcal{A}\{x/n\}
\]

\[
P \models @n \quad \text{iff} \quad n \in \text{fn}(P)
\]

\[
P \models \{\alpha\} \mathcal{A} \quad \text{iff} \quad P \xrightarrow{\alpha} Q \text{ and } Q \models \mathcal{A}
\]
Semantics

\[
\begin{align*}
[\text{True}]_\nu & \triangleq \text{Procs} \\
[\mathcal{A} \land \mathcal{B}]_\nu & \triangleq [\mathcal{A}]_\nu \cap [\mathcal{B}]_\nu \\
[\neg \mathcal{A}]_\nu & \triangleq \text{Procs} \setminus [\mathcal{A}]_\nu \\
[m = n]_\nu & \triangleq \text{if } m = n \text{ then Procs else } \emptyset \\
[0]_\nu & \triangleq \{ P \mid P \equiv 0 \} \\
[\mathcal{A} \mid \mathcal{B}]_\nu & \triangleq \{ P \mid \exists Q, R. P \equiv Q \land R \land Q \in [\mathcal{A}]_\nu \land R \in [\mathcal{B}]_\nu \} \\
[@n]_\nu & \triangleq \{ P \mid n \in \text{fn}(P) \} \\
[Hx.\mathcal{A}]_\nu & \triangleq \{ P \mid \exists Q. P \equiv (\forall n)Q \land n \notin \text{fn}_\nu(\mathcal{A}) \land Q \in [\mathcal{A}]_\nu \} \\
[(\alpha)\mathcal{A}]_\nu & \triangleq \{ P \mid \exists Q. P \xrightarrow{\alpha} Q \land Q \in [\mathcal{A}]_\nu \} \\
[\forall x.\mathcal{A}]_\nu & \triangleq \cap n \in \text{Names.} [\mathcal{A}\{x/n\}]_\nu \\
[X]_\nu & \triangleq \nu(X) \\
[vX.\mathcal{A}]_\nu & \triangleq \{ \psi \subseteq \text{Procs} \mid \psi \subseteq [\mathcal{A}]_\nu[X \leftarrow \psi] \}
\end{align*}
\]
Simple Examples

- **A holds somewhere:**
  \[ ?A \triangleq A \ | \ True \]

- **A holds everywhere:**
  \[ !A \triangleq \neg (\neg A \ | \ True) \]

- **Has exactly one thread:**
  \[ 1 \triangleq \neg 0 \land \neg (\neg 0 \ | \ \neg 0) \]
  \[ ?(1 \land A) \quad A \text{ holds of some thread} \]
  \[ A^* \triangleq ! (1 \Rightarrow A) \quad A \text{ holds of every thread} \]

- **Arithmetic constraints on the number of threads**
  \[ \text{gt}(n) \triangleq \]

- **After an arbitrary step:**
  \[ \Box A \triangleq [\tau]A \lor [?]A \lor [!]A \]

- **Always in the future:**
  \[ \Box^* A \triangleq \forall X. (A \land \Box X) \]
Simple Examples

- **Uses an hidden name** \( x \), that satisfies \( P(x) \)
  
  \[
  H x. \ ( P(x) \land \lnot \diamond x )
  \]

  N.B.: \( \forall x. \mathcal{A} \iff H x. \ ( \mathcal{A} \land \lnot \diamond x ) \)

- Creating bonds through hidden names:

  Let \( P \triangleq ( (\forall n) \ m!(n).n?q(p).Q ) | m?q(q).q!(q).R \)

  Then \( P \models (\lnot \ 0 | \lnot \ 0 ) \land \{ \tau \} 1 \)

  and \( P \models \Diamond H x. ( \Diamond x | \Diamond x ) \)

  \( \Diamond \mathcal{A} \triangleq \{ \tau \} \mathcal{A} \)

- **Keeps no secrets:**

  Public \( \triangleq \lnot H x. \Diamond x \)

- **A holds inside** (insider knows all secrets, but does not tell):

  \( \text{inside}(A) \triangleq \mu X. ((\text{Public} \land A) \lor H x. ( \Diamond x \land X)) \)

  N.B.: \( P \models \lnot \text{inside}(A) \) *iff* \( P \models \text{inside}(\lnot A) \)
The Freshness Quantifier (cf. Gabbay-Pitts)

The freshness quantifier $\forall x.\mathcal{A}$ is defined such that a process $P$ satisfies $\forall x.\mathcal{A}$ if and only if $P$ satisfies $\mathcal{A}\{x/n\}$ for some name $n$ fresh in $P$ and in $\mathcal{A}$.

$P \models_v \forall x.\mathcal{A}$ if and only if $P$ satisfies $\exists n \in \mathcal{N}. n \notin \text{fn}^v(P, \mathcal{A})$ and $P \models_v \mathcal{A}\{x/n\}$

$P \models_v \forall x.\mathcal{A}$ if and only if $\forall n \in \mathcal{N}. n \notin \text{fn}^v(P, \mathcal{A})$ implies $P \models_v \mathcal{A}\{x/n\}$

(A property true of some fresh name is true of any fresh name)

Some properties of the fresh name quantifier:

$\forall x.\mathcal{A} \vdash \forall x.\mathcal{A} \vdash \exists x.\mathcal{A}$

$\neg \forall x.\mathcal{A} \vdash \neg \forall x.\mathcal{A}$

$\forall x. (\mathcal{A} \vdash \mathcal{B}) \vdash \forall x. (\mathcal{A} \vdash \mathcal{B})$

$\forall x.\mathcal{A} \vdash \mathcal{B} \vdash \mathcal{B}$
Some Properties of $\text{H}_x.A$

- "Irrelevant" hidden name quantification reduces to freshness quantification:

$$\text{H}_x.(\mathcal{A} \land \neg \mathcal{O}_x) \vdash \text{H}x. \mathcal{A}$$

- Logical characterisations of scope extrusion:

$$(\text{H}_x.\mathcal{A}) \mid B \vdash \text{H}_x. (\mathcal{A} \mid B \land \neg \mathcal{O}_x)$$
$$(\text{H}_x.\mathcal{A}) \mid (\forall x. B) \vdash \text{H}_x.(\mathcal{A} \mid B)$$

- Some "inversion" principles:

$$\text{H}_x.\emptyset \mathcal{A} \vdash \emptyset \text{H}_x.\mathcal{A}$$
$$(\text{H}_x.\mathcal{A}) \land (\text{H}_x.\mathcal{B}) \vdash \text{H}_x.(\mathcal{A} \land \mathcal{B}) \lor \text{H}_x.\text{H}_y.(\mathcal{A} \land \mathcal{B}\{x/y\})$$
Resource Control and Secrecy

- **Spatial implication**
  \[ \mathcal{A} \rightarrow \mathcal{B} \equiv (\neg \mathcal{A}) \lor \mathcal{B} \]

- **Unique handling of requests**
  \[ \square^* ( \text{inside} \neg \exists y. (\exists x. (y?(x) \land \text{True}) \lor \exists x. (y?(x) \land \text{True}))) \]

- **Resource control (race freedom):**
  \[ \square^* ( \text{inside} \neg \exists y. (\exists x. (y!(x) \land \text{True}) \lor \exists x. (y!(x) \land \text{True}))) \]

- **Secrecy:**
  \[ \square^* (\neg \exists y. Hx.(\mathcal{A}(x) \land (y!(x) \land \text{True}))) \]

\[ \mathcal{A}(x) \equiv \exists y. x(y).\text{True} \quad \text{(never leaks private resources)} \]
“It is always possible for any site to eventually acquire exclusive access to the resource”
“It is always possible for any site to eventually acquire exclusive access to the resource”

\[
\begin{align*}
\text{beh}(n) & \triangleq \ldots \\
\text{node}(n) & \triangleq 1 \land \text{beh}(n) \\
\text{owns}(n, x) & \triangleq \text{node}(n) \land \text{©} x \\
\text{exclusive}(n, x) & \triangleq ( \text{owns}(n, x) \mid \neg \text{©} x ) \\
\text{live} & \triangleq \text{H}x. \text{inside}(\text{obj}(x)\mid \forall n. ?\text{node}(n) \Rightarrow \text{eventually (exclusive}(n, x))) \\
\text{Safety} & \triangleq \text{always ( live )}
\end{align*}
\]
Minimal contextual observations (cf., labeled transitions) are not that easy to define in general.

The composition adjunct operator, introduced in the Ambient Logic [CG00], allows context dependent properties to be defined in a general way (cf., barbed equivalence).

The composition adjunct (guarantee)

\[ \mathcal{A} \mid \mathcal{B} \quad \text{Composition} \]

\[ \mathcal{A} \triangleright \mathcal{B} \quad \text{Guarantee} \]

\[ P \models \mathcal{A} \triangleright \mathcal{B} \iff \forall Q. \text{ if } Q \models \mathcal{A} \text{ then } P \mid Q \models \mathcal{B} \]

The logical equivalence induced by a spatial logic containing just the adjunct operators (and not the “basic” spatial operators) is strong bisimilarity [H04] (on finite processes).
Specification of a Simple Protocol

Client $\triangleq$ $H_x. (\text{Auth}(x) \mid \text{Request}(x))$

Server $\triangleq$ $\forall Y. \forall x. \text{Auth}(x) \triangleright \Diamond (\text{Handler}(x) \mid Y)$

Auth$(x) \triangleq$ ... specification of the authentication protocol ...

By unfolding we get:

Server $\vdash \forall x. \text{Auth}(x) \triangleright \Diamond (\text{Handler}(x) \mid \text{Server})$

We can then prove:

Server $\mid \text{Client} \vdash \Diamond (\text{Server} \mid H_x.(\text{Handler}(x) \mid \text{Request}(x))))$
Expressiveness of Adjunct

- Adjunct allows an internal definition of validity [CG00]
  
  \[ \text{valid}(\mathcal{A}) \triangleq (\neg \mathcal{A}) \triangleright \text{False} \]
  \[ \text{satisfiable}(\mathcal{A}) \triangleq \mathcal{A} \triangleright \text{True} \]
  
  We have: \[ P \models \text{valid}(\mathcal{A}) \iff \forall Q. \ Q \models \mathcal{A} \]

- Validity and model-checking of static quantifier-free spatial logics with adjunct is decidable [CYOH01,CCG03].

- Composition adjunct does not add to the expressiveness of static quantifier free spatial logics [L03] (it does in most other fragments).

- Validity and model-checking of spatial logics with adjunct and existential name quantifier is undecidable [CT02].

- Validity and model-checking of spatial logics with adjunct and hidden name quantifier is undecidable [CG04].

- Validity and model-checking of spatial logics with adjunct and nextstep is undecidable [CL04] (can encode first-order logic).
## Behavioral vs. Spatial Observations

- Basic behavioral and spatial observations look quite elementary. However, the combination of behavioral and spatial properties turns out to be very expressive.

- Behavioral observations are definable in pure spatial logics, exploiting the adjunct [S01,CC01,HLS03,H04].

- Logics with spatial observations and adjunct:
  - More convenient for compositional reasoning.
  - Model and validity-checking is undecidable and incomplete.

- Logics with behavioral and spatial observations:
  - Still expressive...
  - Model-checking is decidable and complete [C04].
Model-Checking

Main technical issues:
- Handling name creation and freshness
- Recursion in the presence of freshness
- Operations on Processes w.r.t. structural congruence

Decidability of Structural Congruence.
For all processes $P$ and $Q$, we can decide whether $Q \equiv P$.
For any finite set of names $M$, we can decide whether $Q \equiv_M P$.

Key to the Completeness Proof.
A coinductive characterization of extended structural congruence (cf. spatial bisimulation).
Model-Checker

MCheck: \textit{Procs} \times \textit{SVal} \times \Phi \rightarrow \text{bool}

MCheck \( (P, v, A) \) assumes \( \text{Dom}(v) \subseteq fpv(A) \)

\begin{align*}
MCheck(P, v, T) & \triangleq \text{true} \\
MCheck(P, v, \neg A) & \triangleq \text{not} \ MCheck(P, v, A) \\
MCheck(P, v, A \land B) & \triangleq MCheck(P, v, A) \land MCheck(P, v, B) \\
MCheck(P, v, 0) & \triangleq \text{Check}(P \equiv 0) \\
MCheck(P, v, A | B) & \triangleq \text{Exists } (Q,R) \in \text{Comp}(P). \\
& \quad MCheck(Q, v, A) \land MCheck(R, v, B) \\
MCheck(P, v, \@n) & \triangleq \text{Check}(\text{Res}(P, n) \neq \emptyset) \\
MCheck(P, v, \langle \alpha \rangle A) & \triangleq \text{Exists } Q \in \text{Comm}(P, \alpha). MCheck(Q, v, A)
\end{align*}
## Model-Checking: Process Observations

### Composition
For every process $P$ we can compute a finite set of pairs of processes $\text{Comp}(P)$ such that:

$$\langle Q, R \rangle \in \text{Comp}(P) \implies P \equiv Q \mid R$$

$$P \equiv Q \mid R \implies \exists \langle Q, R \rangle \in \text{Comp}(P). \ Q \equiv Q' \land R \equiv R'$$

N.B.: $\text{Comp}(P)$ is finite because we don’t have $!P \equiv !P \mid P$.

### Name Restriction
For every process $P$ and name $n$ we can compute a finite set of processes $\text{Res}(P, n)$ such that:

$$Q \in \text{Res}(P, n) \implies P \equiv (\forall n)Q$$

$$P \equiv (\forall n)Q \implies \exists R \in \text{Res}(P, n). \ R \equiv Q$$

N.B.: $\text{Res}(P, n) = \emptyset$ if and only if $n \in \text{fn}(P)$

### Commitment
For every process $P$ and action $\alpha$ we can compute a finite set of processes $\text{Comm}(P, \alpha)$ such that:

$$Q \in \text{Comm}(P, \alpha) \implies P \xrightarrow{\alpha} Q$$

$$P \xrightarrow{\alpha} Q \implies \exists R \in \text{Comm}(P, \alpha). \ R \equiv Q$$
Model-Checking: Syntactic Valuations

**Syntactic Valuation.**

A *syntactic valuation* \((\nu \in SVal)\) is a sequence

\[
[X_1 \rightarrow (S_1, A_1)] \ldots [X_n \rightarrow (S_n, A_n)]
\]

of assignments such that:

- Each \(X_i\) is a propositional variable.
- Each \(A_i\) is a fixpoint formula of the form \(\nu X_i. B_i\).
- \(fpv(A_i) \subseteq \{ X_1, \ldots, X_{i-1} \}\)

**Free Names of formula \(\mathcal{A}\) under syntactic valuation \(\nu\).**

\[
fs^\nu(\mathcal{A}) \triangleq \bigcup \{ fs^\nu(\mathcal{B}) \mid \nu(X) = (S, B) \land X \in fpv(\mathcal{A}) \} \cup fn(\mathcal{A})
\]

**c.f.,** \(fn^\nu(\mathcal{A}) \triangleq \bigcup \{ supp(\nu(X)) \mid X \in fpv(\mathcal{A}) \} \cup fn(\mathcal{A})\)
**Model-Checker**

\[ \text{MCheck: } \text{Procs} \times \text{SVal} \times \Phi \rightarrow \text{bool} \]

\[ \text{MCheck} (P, v, \mathcal{A}) \text{ assumes } \text{Dom}(v) \subseteq \text{fpv}(\mathcal{A}) \]

\[ \text{MCheck} (P, v, \forall x.\mathcal{A}) \triangleq \]

\[ \text{let } M = \text{fs}^v(\forall x.\mathcal{A}) \cup \text{fn}(P) \]

\[ \text{in Forall } n \in M \cup \{ \text{fresh}(M) \}. \text{MCheck} (P, v, \mathcal{A}\{x/n\}) \]

\[ \text{MCheck} (P, v, \text{Hx.}\mathcal{A}) \triangleq \]

\[ \text{let } n = \text{fresh}(\text{fs}^v(\text{Hx.}\mathcal{A})) \text{ and } Q \in \text{Res}(P,n) \]

\[ \text{in MCheck} (Q, v, \mathcal{A}\{x/n\}) \]

**N.B.** \(\text{fresh}(M)\) picks some name out of the finite set \(M\).

**N.B.** Any \(\text{fresh}()\) function can be used (e.g., ad-hoc gensym).
Model-Checker

\[ \text{MCheck}: \text{Procs} \times \text{SVal} \times \Phi \rightarrow \text{bool} \]

\[ \text{MCheck}(P, v, A) \text{ assumes } \text{Dom}(v) \subseteq \text{fpv}(A) \]

\[ \text{MCheck}(P, v, vX.A) \triangleq \text{MCheck}(P, v[ X \rightarrow (\{P\}, vX.A)], A) \]

\[ \text{MCheck}(P, v, X) \triangleq \]

\[ \text{if } \text{In}(P, v, X) \text{ then true} \]

\[ \text{else let } (S, vX.G) = v(X) \]

\[ \text{in } \text{MCheck}(P, v[ X \rightarrow (S \cup \{P\}, vX.G)], G) \]

\[ \text{In}(P, v, X) \triangleq \text{let } (S, F) = v(X) \]

\[ \text{and } M = \text{fs}^v(F) \]

\[ \text{in Exists } Q \in S. Q \equiv_M P \]
Soundness of the Model-Checker

**Semantic valuation.** For any (syntactic) valuation \( \nu = w[X \to (S, \nu X.\bar{A})] \) we define a corresponding (semantic) valuation \( \nu^* \) by letting

\[
\nu^*(X) \triangleq \text{Gfix}(\lambda U. \text{Clos}(S, M) \cup [\bar{A}]_{w^*[X \leftarrow \nu]} )
\]

wherever \( \nu(X) = (S, \nu X.\bar{A}) \) and \( M = fs^\nu(\nu X.\bar{A}) \)

**Soundness.** If \( \text{MCheck}(P, \nu, \bar{A}) = \text{true} \) then \( P \in [\bar{A}]_{\nu^*} \)

If \( \text{MCheck}(P, \nu, \bar{A}) = \text{false} \) then \( P \notin [\bar{A}]_{\nu^*} \)

**Key to the Proof.**

Support & Closure.

\[
\text{supp}( [\bar{A}]_{\nu^*} ) \subseteq \text{fn}^\nu( \bar{A} ) \subseteq \text{fs}^\nu( \bar{A} )
\]

If \( P \in [\bar{A}]_\nu \) and \( \text{fn}^\nu( \bar{A} ) \subseteq M \) and \( P \equiv_M Q \) then \( Q \in [\bar{A}]_\nu \)

Kozen-Winskel.

Define the mapping \( \phi \) such that \( \phi(S) \triangleq [\bar{A}]_{\nu[X \leftarrow S]} \). Then:

For all \( \psi \subseteq \text{Procs} \), \( \psi \subseteq \text{Gfix}(\phi) \) iff \( \psi \subseteq \phi(\text{Gfix}(\lambda S.\phi(\psi \cup S))) \)
Completeness of the Model-Checker

Reachability.
\[ P \in \text{Reach}(P) \]
\[ P \in \text{Reach}(P), Q \in \text{Com}(P, \alpha) \Rightarrow Q \in \text{Reach}(P) \]
\[ P \in \text{Reach}(P), (Q, R) \in \text{Comp}(P) \Rightarrow Q, R \in \text{Reach}(P) \]
\[ P \in \text{Reach}(P), Q \in \text{Res}(P, n) \Rightarrow Q \in \text{Reach}(P) \]

Bounded. A process \( P \) is bounded if for every finite set of names \( M \), the set of equivalence classes \( \text{Reach}(P) / \equiv_M \) is finite.

N.B.: All recursion-free processes are bounded.
All finite-control processes are bounded.
It is not the case that every terminating process is bounded.

Completeness & Decidability.
If \( P \) is bounded and \( P \in \left[ A \right]_{\nu} \), then \( \text{MCheck}(P, \nu, A) = \text{true} \)
The Spatial Logic Model Checker [VC04,05,06]

- Developed in UNLisbon, based on the techniques described above [Cai04] (complete for “bounded” processes).
- On-the-fly state-space generation.
- Ocaml implementation (available in source form).
- Supports the full $\pi$-calculus with parametric recursion.
- Supports a full adjunct-free logic with parametric recursion.
- Version 1.1 is available on the web:
  
  http://ctp.di.fct.unl.pt/SLMC/

- You may find some worked out examples there.
Expressiveness and Intensionality
The logical equivalence induced by spatial logics is quite sensitive to the presence of particular process operators, logical operators, and to the presentation of structural congruence. E.g.,

Sangiorgi has shown, in seminal work [San01], that for the ambient logic and finite public ambient calculus, $=_{L}$ coincides with $\equiv$ (intensionality).

If we consider the choice-free finite $\pi$-calculus and the core spatial logic, we also obtain $\equiv = =_{L}$.

However, if we add choice or recursion, then we have $\equiv \subsetneq =_{L}$.

In a certain distributed calculus (below), if the semantics is crafted so that a single site may fail (at each reduction step) instead of an arbitrary subsystem, then strong bisimilarity collapses to spatial bisimilarity.

It is then important to study “principled” spatial models and spatial logics, for which logical equivalence coincides with isomorphism of (the intended / observable) spatial structure.
Characterisation of $=_{L}$ in the $\pi$-calculus

- Logical Equivalence of Processes ($=_{L}$)

\[ P =_{L} Q \triangleq \forall A. (P \vdash A \iff Q \vdash A) \]

- Axiomatization of logical equivalence on the full pi-calculus with choice and recursion [Cai04]:

$=_{L}$ is the least congruence relation $\equiv^\varepsilon$ on processes such that:

\[
\begin{align*}
    P \equiv Q & \implies P \equiv^\varepsilon Q \\
    P + P & \equiv^\varepsilon P \\
    P \equiv^\varepsilon Q\{Y / (x)P\} & \implies P \equiv^\varepsilon (\text{rec } Y[x].Q)[x]
\end{align*}
\]

We have: Both $\equiv^\varepsilon$ and $=_{L}$ are decidable.

We have: $\equiv \subset =_{L} \subset \vdash$ (strict inclusions)

Spatial Bisimulation ($\pi$-calculus)

- Characterizes indistinguishable states (coinductively)
  
  A binary relation $B \subseteq \mathcal{P} \times \mathcal{P}$ is a spatial bisimulation if

  for all $(s, r) \in B$

  if $s \xrightarrow{\alpha} s'$ then there is $r'$ s.t. $r \xrightarrow{\alpha} r'$ and $(s', r') \in B$
  (and conversely)

  if $s \equiv 0$ then $r \equiv 0$
  (and conversely)

  if $s \equiv p \mid q$ for some $p, q$ then there are $u, v$ s.t. $r \equiv u \mid v$
  and $(p, u) \in B$ and $(p, u) \in B$
  (and conversely)

  if $s \equiv (\nu n)p$ for some $p, n$ exists $u$ s.t. $r \equiv (\nu n)u$ and $(p, u) \in B$
  (and conversely)

- Spatial bisimilarity $\sim_s^s :$ the greatest spatial bisimulation

  $\sim_{s0} = \bigcup \{ B \mid B \text{ is a spatial bisimulation} \}$
Spatial Bisimulation ($\pi$-calculus)

- We have
  \[
  n!() \mid m!() \not\simeq^s n!().m!()+m!().n!()
  \]
  although
  \[
  n!() \mid m!() \sim n!().m!()+m!().n!()
  \]
  (cf. “true concurrency” semantics)

- We have
  \[
  0 \not\simeq^s (\forall n)n!()
  \]
  although
  \[
  0 \sim (\forall n)n!()
  \]
  ($\not\simeq^s$ distinguishes deadlock from proper termination)

- If $P,Q$ are sequential (no parallel composition) and public (no restricted names) then $P \sim Q$ implies $P \sim^s Q$
In general, we may expect spatial observations, as captured by a spatial logic SL, to induce a degree of intensionality, in the sense that logical equivalence is strictly finer than behavioral equivalence.

However, purely behavioral equivalences in process calculi with spatial constructs (e.g., mobile ambients) are already fairly sensitive to system properties usually considered “intensional”, such as arithmetical constraints in the number of sites, etc ...

For example, in Sangiorgi [S01] has shown that for the public ambient calculus and ambient logic,\
\[ \equiv (\Rightarrow) =_{\text{SL}} (\subset) \approx \]

But how far is \( =_{\text{SL}} \) from ~ (and \( \approx \))?

There are natural models where \( =_{\text{SL}} (\Rightarrow) \approx \)
A minimal distributed model

A set $A$ of actions $(\alpha, \tau)$ where $\alpha = n$ or $\alpha = \overline{n}$

A set $\mathcal{P}$ of processes

\[
P, Q, R ::= \text{nil} | P | Q | \alpha.P | \text{go}.P
\]

A set $\mathcal{N}$ of networks

\[
N, M, S ::= 0 | [P] | M | N
\]

Structural congruence $\equiv$

\[
\begin{align*}
P | \text{nil} & \equiv P \\
P | Q & \equiv Q | P \\
(P | Q) | R & \equiv P | (Q | R) \\
(N | M) | S & \equiv N | (M | S)
\end{align*}
\]

\[
P \equiv Q \\
\hline
[ P ] \equiv [ Q ]
\]
Extensionality in Spatial Observations

A minimal distributed model

A set \( A \) of actions \((\alpha, \tau)\) where \( \alpha = n \) or \( \alpha = \bar{n} \)

A set \( \mathcal{P} \) of processes

\[
P, Q, R ::= \text{nil} \mid P \mid Q \mid \alpha.P \mid \text{go}.P
\]

A set \( \mathcal{N} \) of networks

\[
N, M, S ::= 0 \mid [ P ] \mid M \mid N
\]

Reduction \( \rightarrow \)

\[
[ \alpha.P \mid \bar{\alpha}.Q \mid R ] \rightarrow [ P \mid Q \mid R ] \quad M \equiv M' \quad M' \rightarrow N' \quad N' \equiv N
\]

\[
[ \tau.P \mid R ] \rightarrow [ P \mid R ] \quad M \rightarrow N
\]

\[
[ \text{go}.P \mid R ] \mid [ Q ] \rightarrow [ R ] \mid [ P \mid Q ] \quad M \rightarrow N
\]

\[
[ P ] \mid N \rightarrow 0 \quad M \mid S \rightarrow N \mid S
\]
A minimal distributed model

A set $A$ of actions $(\alpha, \tau)$ where $\alpha = n$ or $\alpha = \overline{n}$

A set $P$ of processes

$$P, Q, R ::= \text{nil} | P | Q | \alpha.P | \text{go}.P$$

A set $N$ of networks

$$N, M, S ::= 0 | [P] | M | N$$

Reduction $\rightarrow$

- $[\alpha.P | \overline{\alpha}.Q | R] \rightarrow [P | Q | R]$ (migration)
- $[\tau.P | R] \rightarrow [P | R]$
- $[	ext{go}.P | R] | [Q] \rightarrow [R] | [P | Q]$
- $[P] | N \rightarrow 0$ (failure)

$M \equiv M' \quad M' \rightarrow N' \quad N' \equiv N$

$M \rightarrow N$

$M | S \rightarrow N | S$
A minimal distributed model

A set \( A \) of actions \((\alpha, \tau)\) where \( \alpha = n \) or \( \alpha = \overline{n} \)

A set \( \mathcal{P} \) of processes

\[
P, Q, R ::= \text{nil} \mid P \mid Q \mid \alpha.P \mid \text{go}.P
\]

A set \( \mathcal{N} \) of networks

\[
N, M, S ::= 0 \mid [P] \mid M \mid N
\]

Barb observation

\[
[\alpha.P \mid Q] \Downarrow \alpha
\]

We then pick observational equivalence for networks (undistinguishability) to be strong reduction barbed congruence \( \simeq \) defined in the standard way.
A minimal distributed model

A set $A$ of actions $(\alpha, \tau)$ where $\alpha = n$ or $\alpha = \overline{n}$

A set $P$ of processes

$$P, Q, R ::= \text{nil} \mid P \mid Q \mid \alpha.P \mid \text{go}.P$$

A set $N$ of networks

$$N, M, S ::= 0 \mid [P] \mid M \mid N$$

Labels

$$\lambda ::= \tau \mid \alpha \mid \overline{\alpha} \mid [\alpha]$$

Labeled observations $\overset{\lambda}{\rightarrow}$

$$[\overline{\alpha}.P \mid Q] \overset{\alpha}{\rightarrow} [P \mid Q]$$

$$[\alpha.P \mid Q] \overset{\alpha}{\rightarrow} [P \mid Q]$$

$$N \overset{[\alpha]}{\rightarrow} N \mid [\alpha.\text{nil}]$$
Extensionality in Spatial Observations

- **Minimal Spatial Logic**

\[ \mathcal{A}, \mathcal{B}, \mathcal{C} ::= \text{True} \mid \mathcal{A} \land \mathcal{B} \mid \neg \mathcal{A} \mid 0 \mid \mathcal{A} \mid \mathcal{B} \mid (\lambda)\mathcal{A} \]

- **Theorem (Characterization) [CaiVie06]**

\[ \mathcal{M} \sim \mathcal{N} \text{ if and only if } \mathcal{M} =_{L} \mathcal{N} \]

- **Theorem (Minimality)**

In the model considered, no proper fragment of the minimal spatial logic preserves its separation power.

- **Moral**

Spatial observations are not necessarily intensional or arbitrary. In distributed systems, computational contexts can already “see” some spatial structure, due to migration, failures, differences in communication latency, and other phenomena...
Proof Systems for Spatial Logics
A Proof System for Spatial Logic

We define a labeled sequent calculus where labels denote \(\pi\)-calculus processes and accessibility is reduction:

\[
\{ S \} \ u_1 : A_1, \ldots, u_n : A_n \vdash v_1 : B_1, \ldots, v_m : B_m
\]

\(A_i, B_j\) are formulas
\(u_i, v_j\), labels are indexes, elements of

the term \(\pi\)-algebra \(P = \langle N, I, 0, |, \nu, \leftrightarrow_N, \leftrightarrow_I \rangle\) over process

variables \(X\), where \(N\) are name terms, and \(I\) are process terms.

\(S\) is a finite set of constraints, describing the “current world”

Constraints are either:

Equations \(u = v\) between indexes (to handle spatial structure)

Distinctions \(n \# m\) (to handle freshness)

Reductions \(u \rightarrow v\) (to handle dynamics)
A Proof System for Spatial Logic

- **Closure axioms for constraint sets, e.g.,**
  \[(\forall n)(u \mid (\forall n)v) \vdash_s ((\forall n)u) \mid (\forall n)v\]
  \[n \neq p, m \neq p \Rightarrow (m \leftrightarrow n)p \vdash_s p\]

- **Propositional Rules, e.g.:**
  \[\langle S \rangle \; ^{\wedge L} \; \Gamma, u : \mathbb{A}, u : \mathbb{B} \vdash_\Delta \]
  \[\langle S \rangle \; ^{\wedge R} \; \Gamma \vdash u : \mathbb{A}, \Delta \quad \langle S \rangle \; ^{\wedge R} \; \Gamma \vdash u : \mathbb{B}, \Delta \]
  \[\langle S \rangle \; ^{\wedge R} \; \Gamma \vdash u : \mathbb{A} \land \mathbb{B}, \Delta\]

- **Spatial Rules, e.g.:**
  \[(\mid L) \; \not x, \not y \; \text{not free in the conclusion} \]
  \[\langle S \rangle \; ^{\mid L} \; u \equiv X \mid Y \; \Gamma, Y : \mathbb{A}, \; X : \mathbb{B} \vdash_\Delta \]
  \[\langle S \rangle \; ^{\mid R} \; \Gamma \vdash v : \mathbb{A}, \Delta \quad \langle S \rangle \; ^{\mid R} \; \Gamma \vdash t : \mathbb{B}, \Delta \quad u \equiv_s v \mid t\]
  \[\langle S \rangle \; ^{\mid R} \; \Gamma \vdash u : \mathbb{A} \mid \mathbb{B}, \Delta\]

- **World Rules, e.g.:**
  \[\langle S, u \equiv 0 \rangle \; ^{\#} \; \Gamma \vdash_\Delta \]
  \[\langle S, x \neq N, u \equiv (\forall x)Y \rangle \; ^{\#} \; \Gamma \vdash_\Delta\]

- **Freshness Rules, e.g.:**
A Simple Proof

\[ \text{(0 R)} \quad u \equiv Z_0 \quad \frac{\vdash \Gamma}{\Gamma \vdash u : 0, \Delta} \]

\[ \text{(| R)} \quad \Gamma \vdash \nu : A, \Delta \quad \Gamma \vdash t : B, \Delta \quad u \equiv \nu | t \]

\[ \text{(0 L)} \quad \frac{\langle S, u \equiv 0 \rangle \Gamma \vdash \Delta}{\langle S \rangle \Gamma, u : 0 \vdash \Delta} \]

\[ \text{(| L)} \quad x, \gamma \text{ not free in the conclusion} \]

\[ \frac{\langle S, u \equiv X | \gamma \rangle \Gamma, X : A, \gamma : B \vdash \Delta}{\langle S \rangle \Gamma, u : A | B \vdash \Delta} \]

5\: \langle Z \equiv X | \gamma, Z \equiv 0, X \equiv 0 \rangle X : A, \gamma : B \vdash Z : A \quad \text{(Id) since } z = x

4\: \langle Z \equiv X | \gamma, Z \equiv 0 \rangle X : A, \gamma : B \vdash Z : A \quad 5, (S \ | \ 0) \text{ since } x \ | \ y = 0

3\: \langle Z \equiv X | \gamma \rangle X : A, \gamma : B, Z : 0 \vdash Z : A \quad 4, (0 \ L)

2\: \langle \rangle Z : A | B, Z : 0 \vdash Z : A \quad 3, (| L)

1\: \langle \rangle Z : (A | B) \land 0 \vdash Z : A \quad 2, (\land \ L)
Reasoning with Spatial Logic
“It is always possible for any site to eventually acquire exclusive access to the resource”
Specifications for Directory Nodes

\[ \text{idle}(f, m, l) \iff 1 \land \forall \alpha. [ \alpha ] \exists a, n. ( \alpha = f^?(a, n) \land ( \text{idle}(f, m, n) \mid l!(a, f) ) \lor ( \alpha = \tau \land \text{twaiter}(f, m) \mid l!(m, f) ) ) \]

\[ \text{towner}(f, m) \iff 1 \land \forall \alpha. [ \alpha ] \exists a, n. ( \alpha = f^?(a, n) \land \text{owner}(f, m, n, a) ) \]

\[ \text{owner}(f, m, l, q) \iff \ldots \]

\[ \text{twaiter}(f, m) \iff \ldots \]

\[ \text{waiter}(f, m, l, q) \iff \ldots \]

\[ \text{node}(f) \iff \exists m, l, q. \text{idle}(f, m, l) \lor \text{twaiter}(f, m) \lor \text{towner}(f, m) \lor \text{owner}(f, m, l, q) \lor \text{waiter}(f, m, l, q) \]
Specifications for the Directory

\[ \begin{align*}
\text{BootState} & \triangleq \exists r, u. ( \text{towner}(f, u) \mid \\
& \quad (\exists f, m, l. \text{idle}(f, m, l))^* ) \\
& \quad \land \forall n. (\text{?node}(n) \Rightarrow \text{?Path}(n, r))
\end{align*} \]

\[ \text{System} \triangleq (\exists n. \text{node}(n) \lor \text{Msg})^* \]

\[ \text{Link}(a, b) \triangleq \exists c. b!(c, a) \lor \text{LinkedNode}(a, b) \]

\[ \text{Path}(a, b) \iff (a = b) \lor \exists c. (\text{Link}(a, c) \lor \text{Link}(c, a) \mid \text{Path}(c, b)) \]

\[ \text{UniquePath} \triangleq \neg \exists a, b. (a \neq b) \land (\text{?Path}(a, b) \mid \text{?Path}(a, b)) \]

N.B.: \text{UniquePath} \vdash \text{NoCycle}

\[ \text{Ok} \triangleq \text{UniquePath} \]

\[ \text{NeverLoops} \triangleq \text{always}(\text{Ok}) \]
Plan to show: $\text{Ok} \Rightarrow \square \text{Ok}$

Assume: $\text{Ok} \land \text{System}$

Then prove: $(\Diamond \neg \text{Ok}) \Rightarrow \text{False}$

Assume: $\Diamond \neg \text{Ok}$

Use the rule: $\Diamond \exists \mathcal{X}, \, m. \, \big( \, m \mid \mathcal{X} \mid 1 \land \{ \, m \, \} \, \big( \, \mathcal{X} \triangleright \mathcal{A} \, \big) \, \big)$

$\exists \mathcal{X}, \, m. \, \big( \, m \land \text{Msg} \big) \mid \big( \, \mathcal{X} \land \text{Ok} \big) \mid \big( \, \text{Node} \land \{ \, m \, \} \, \big( \, \mathcal{X} \triangleright \neg \text{Ok} \, \big) \, \big)$

Then, proceed by case analysis on $\text{Node} \triangleq \exists \, n. \, \text{node}(n)$

$\exists \mathcal{X}. \, \Diamond \, \big( \, \mathcal{X} \land \text{Ok} \mid \big( \, \text{Post} \land \big( \, \mathcal{X} \triangleright \neg \text{Ok} \big) \, \big) \, \big)$

$\text{Ok} \mid \text{Post} \Rightarrow \text{Ok}$

$\Diamond \, (\, \text{Ok} \land \neg \text{Ok} \,)$

$\Diamond \, \text{False}$

$\text{False}$

$\text{Msg} \Rightarrow \exists f, \, m, l. \, f!(m, l) \lor \exists m. \, m!()$

$\text{Node} \Rightarrow [ \, m \, ] \, \text{Post}$
Proving a Safety Property

Details of one case (towner(v, n)):

∃ X, m. (m | (X ∧ Ok) | (Node ∧ {m} (X ▷ ¬Ok)))

∃ X, m, v, n. (m | (X ∧ Ok) | (towner(v, n) ∧ {m} (X ▷ ¬Ok)))

∃ m, v, u. (v!(m, u) | (X ∧ Ok) | (towner(v, n) ∧ {v?(m, u)} (X ▷ ¬Ok)))

∃ m, v, u, v! (m, u) | (X ∧ Ok ∧ ¬Path(v, u)) | (towner(v, n) ∧

{ v?(m, u) } (X ▷ ¬Ok))

◊ (X ∧ Ok ∧ ¬Path(v, u) | (owner(v, n, u, m) ∧ (X ▷ ¬Ok))

Ok ∧ ¬Path(a, b) ⇒ (Path(a, b) ▷ Ok) \hspace{1cm} \text{(Auxiliary Lemma)}

Ok ∧ ¬Path(v, u) | owner(v, n, u, m) ⇒ Ok

◊ (Ok ∧ ¬Ok)

◊ False

False
A property $A$ is **distributed** if the following entailments hold:

- $A \mid A \vdash A$
- $A \vdash \text{True} \Rightarrow A$

For example, $A^*$ is distributed for all $A$.

In our example, property **UniquePath** is distributed.

Then, the following proof rule is sound for any distributed $G$:

$$S \vdash ( \bigvee_i C_i )^*$$

$$C_i \vdash \forall \alpha. [\alpha] \bigvee_j ( \alpha = m_j \land P_j )$$

$$\bigwedge_j G \Rightarrow ( m_j \Rightarrow ( P_j \triangleright G ) )$$

$$S \land G \vdash \Box G$$
A Spatial Type System for Services
Some Key Features of Distributed Services

**Distributed Services**
- Also known as distributed objects, but ...
- On-the-fly system composition
- Services depend (call) on other (remote) services
- Tasks must be properly scheduled (workflow)
- Services may be dynamically bound and (ref) passed around

**Procedural Abstraction for Tasks**
- Using distributed remote procedure call mechanisms
- Control flow involves distributed (long term) transactions

**Coordination Abstractions**
- Parallel Composition of Tasks
  - Tasks are independent when never get to compete for resources
  - Independent tasks appear to run “simultaneously” (no interference)
  - This is the default behavior of the “global computer”
- Sequential Composition of Tasks
  - Causality, data flow, and resource competition leads to sequentiality
Some Key Features of Distributed Services

**Resources as disciplined objects**

- Resources are (at least partially) unshareable services/objects
  - E.g., a file, a session, a key, a physical device, ...
- Must be used according to a strict discipline / protocol (otherwise faults may occur)

**Control of Resources**

- Resources may be passed around, buffered in pools, etc, ...
- In principle, at any given moment the sets of resources usable by each one of the ongoing tasks should be kept disjoint (cf., the separation principle).
- However, resource sharing policies may be of fine granularity (think of “multiple readers-unique writer” as a special case)

**Spatial-Behavioral Types**

- We develop a core programming language and a compositional type system to discipline interactions and resource usage on distributed services systems, inspired by spatial logic.
A General Type Structure for Services

- $U \mid V$
  Independent Tasks (Spatial Decomposition)
- $U \& V$
  Optional Tasks
- $U ; V$
  Sequential Tasks
- $T^0$
  Owned Task
- $U \triangleright V$
  Guarantee (Spatial Compositionality)

Spatial Logic Semantics of Types

$P \models T$ (logical satisfaction, compositionally defined)

Types denote “properties of processes”, not “processes”
The semantics of types entails the intended safety properties
Composition with Spatial Types

\[ \text{backsys} : T(bk) \triangleright T(gw) \triangleright T(f) \triangleright T(h) \]

\[ \text{travel} : 0 \triangleright T(f) \]

\[ \text{gateway} : T(bk) \triangleright T(gw) \]

\[ \text{bank} \]

\[ \text{debit()} = \]

\[ \text{pay(s)} = \]

\[ \text{if } bk\text{.debit()} \]

\[ \text{then } s\text{.book()} \]

\[ \text{else } s\text{.free()} \]

\[ \text{broker} \]

\[ br : \text{Rec } \chi \text{. (flight() | hotel())}; \text{order(); } \chi \]

\[ f : \text{Rec } \chi \text{. flight();(book(); } \chi \text{ & free(); } \chi) \]

\[ h : \text{Rec } \chi \text{. hotel();(book(); } \chi \text{ & free(); } \chi) \]

\[ gw : \text{Rec } \chi \text{. pay(book() & free()); } \chi \]

\[ \text{TravelSvc} : T(bk) \triangleright T(br) \]

\[ \text{broker} : (T(gw) \triangleright T(f)) \triangleright (T(h) \triangleright T(br)) \]
Each of the service calls `br.hotel()` and `br.flight()` is handled by (spatially) separated parts of the system.
Reflecting Spatial Decomposition on Types

The semantics of the \((U \mid V)\) typing implies that the services \(br\.hotel()\) and \(br\.flight()\) may be (safely) used concurrently.
Composing BackSys and broker

\[ \text{bank: } 0 \triangleright T( b ) \]

\[ \text{travel: } 0 \triangleright T( f ) \]

\[ \text{gateway: } T( bk ) \triangleright T( gw ) \]

\[ \text{bank: } \text{debit()} = \]
\[ \text{pay(s) = if } bk \text{.debit()} \text{ then s.book() else s.free()} \]

\[ \text{travel: } \text{flight}() = f \text{.flight()} ; \]
\[ \text{hotel()} = h \text{.hotel()} ; \]
\[ \text{order()} = gw \text{.pay}(f) ; gw \text{.pay}(h) ; \]

\[ \text{gateway: } \text{gw} = \text{Rec } \chi . \text{pay(book() & free());} \chi \]

\[ \text{broker: } \text{br} = \text{Rec } \chi . (\text{flight() | hotel()}; \text{order();} \chi \]

\[ \text{BackSys: } 0 \triangleright T( gw ) | T( f ) | T( h ) \]

\[ \text{broker: } (T( gw ) | T( f ) | T( h )) \triangleright T( br ) \]

\[ \text{TravelSys: } 0 \triangleright T( br ) \]
Another Decomposition

\( f : \text{Rec } X. \ \text{flight;}(\text{book};X \ & \ \text{free};X) \)

\( h : \text{Rec } X. \ \text{hotel;}(\text{book};X \ & \ \text{free};X) \)

\( gw : \text{Rec } X. \ \text{pay(book; & free};X) \)

\[ 
\text{travel: } 0 \triangleright T(f) \\
\text{gateway: } T(bk) \triangleright T(gw) \\
\text{bank: } \text{debit} = | \\
\text{broker: } \text{flight} = f.\text{flight}(); \\
\text{hotel} = h.\text{hotel}(); \\
\text{order} = gw.\text{pay}(f); gw.\text{pay}(h); \\
\text{br} : \text{Rec } X. (\text{flight} | \text{hotel});\text{order};X \\
\text{BackSys: } T(bk) \triangleright T(gw) | T(f) | T(h) \\
\text{TravelSvc: } T(bk) \triangleright T(br) \\
\]
Intended Safety Properties of Typing

**Flow**: $\triangleq$ Available $\land \neg$ Race $\land$ Unique

**Safe**: $\triangleq$ $\square$ Flow

The subsystem responsible for the distributed execution of a given service call should always satisfy the Flow invariant.

```plaintext
travel
broker
gateway
accom

bank

debit() =

pay(s) = if bk.debit() then s.book() else s.free()

flight() = f.flight();
hotel() = h.hotel();
order() = gw.pay(f); gw.pay(h);
```
Intended Safety Properties of Typing

Flow $\triangleq$ Available $\land$ $\neg$Race $\land$ Unique  
Safe $\triangleq$ $\square$Flow

The subsystem responsible for the distributed execution of a given service call should always satisfy the Flow invariant.
The Object Model

- **Objects**
  - Name (unique)
  - Methods (passive code): $f(x) = E$
  - Threads (active code): $c(E)$

- **Systems**
  - A system is composed by several objects, each one possibly running several threads concurrently.
  - Objects may call methods of any other objects they know about.
  - Object names may be passed around (name mobility).
## The Process Calculus

### Expressions

- $n, m, p \in \text{Names}$
- $u, v \in \text{Val} \::=\ \text{Values}$
  - \text{stop} \quad \text{Primitive}
  - $n \quad \text{Name}$

### Variables

- $x, y, z \in \text{Variables}$

### Expressions

- $E, F \in \text{Expr} \::=\ \text{Expressions}$
  - $x \quad \text{Identifier}$
  - $v \quad \text{Value}$
  - $n.1(v) \quad \text{Call}$
  - $n.c() \quad \text{Back}$
  - \text{let } x = E \text{ in } F \quad \text{Composition}$
  - \text{new } [M] \quad \text{Instantiation}$

### Processes

- $P, Q, R \in \text{Proc} \::=\ \text{Processes}$
  - \text{0} \quad \text{Void}$
  - $n[M; T] \quad \text{Object}$
  - $P \mid Q \quad \text{Objects}$

**N.B.:**

- In any object $n[M; T]$
  - Methods in $M$ have **distinct labels**.
  - Threads in $T$ have **distinct names**.

- In any process $P$
  - Objects in $P$ have **distinct names**.
### Types

\[ X, Y, Z \in \text{Type Variables} \]

\[ T, U, V \in \text{Type} ::= \text{Types} \]

- \text{nil} \quad \text{Stop}
- \text{1}(U)V \quad \text{Method}
- \text{c}(m:U)V \quad \text{Thread}
- U \mid V \quad \text{Spatial}
- U ; V \quad \text{Sequential}
- U \& V \quad \text{Conjunction}
- U^\circ \quad \text{Owned}
- X \quad \text{Variable}
- \text{Rec } X.T \quad \text{Recursion}

### Atomic Typing Assertion

\( n : T \) \quad \text{(object n has type T)}

### Typing Contexts \((A, B)\)

\[ n_1 : T_1, \ldots , n_k : T_k \]

\text{N.B. write } A(n_1) = T_1

when \( A \) is \( n_1 : T_1, \ldots , n_k : T_k \)

### Typing Judgment (Expressions).

\[ E :: A \triangleright B [T] \]

### Typing Judgment (Processes).

\[ P :: A \triangleright B \]

### A valid expression judgment.

\[ A \triangleq n: \text{fly();fr()}, \]

\[ h: \text{hot();bk()&fr()}, \]

\[ g: \text{do(bk())} T \]

\[ B \triangleq n: \text{fr()}, \]

\[ h: \text{nil}, \]

\[ g: \text{nil} \]

\[ E \triangleq (n.\text{fly()} | h.\text{hot()}); g.\text{do(h)} \]

\[ E :: A \triangleright B [T] \]
### Subtyping

#### Congruences.

\[ U <: U', V <: V' \Rightarrow U | V <: U' | V' \]
\[ U <: U', V <: V' \Rightarrow U ; V <: U' ; V' \]

*etc...*

#### Conjunction.

\[ U \& V <: U \]
\[ U \& V <: V \]
\[ T <: U, T <: V \Rightarrow T <: U \& V \]

#### Composition.

\[ U | \text{nil} <-: U \]
\[ U | V <-: V | U \]
\[ (U | V) | T <-: U | (V | T) \]

#### Sequential.

\[ U ; \text{nil} <-: U \quad U | V <-: U ; V \]
\[ \text{nil} ; U <-: U \]
\[ (U ; V) ; T <-: U ; (V ; T) \]
\[ (U ; U') | (V ; V') <-: (U | V) ; (U' | V') \]

#### Owned.

\[ U^0 <-: U \quad \text{nil} <-: \text{nil}^0 \]
\[ U^0 <-: \text{nil} \quad U^0 <-: U^{00} \]
\[ (U | V)^0 <-: U^0 | V^0 \]
\[ (U ; V)^0 <-: U ; V^0 \]
\[ U^0 ; V <-: U^0 | V \]

#### Contexts.

\[ V <-: U \Rightarrow n: V <-: n: U' \]
Soundness of Typing

There is an embedding $[\ ]$ of our types into a spatial logic such that:

\[ P :: \Gamma \vdash \Delta. \text{ Then } P \models [\Gamma] \triangleright [\Delta]. \]

Let $P :: \Gamma \vdash A$. Then $P \models [A]$.

In particular, if $P :: \Gamma \vdash A$ is derivable then $P \models \text{Safe}$ holds.

The soundness proof is driven by semantical reasoning (think of $\Gamma \vdash \Delta$ as a logical relation).

It is very natural and modular (we look once at each rule).

The type system is indeed a proof system (in the standard sense) for satisfaction ($P \models A$) w.r.t. the underlying logic.
State and Resource Control (Example)

R ⊆ use(*
PoolType ⊆ !( free(R°) & alloc(R )
pool : PoolType

server [
  init() = s!(stop);
  open() =
    let x = pool.alloc() in s!(x)
  use() =
    let x = s? in ( x.use() ; s!(x) )
  close() =
    let x = s?
    in ( pool.free(x); s!(stop); )
]

ResType ⊆ use(*
SrvType ⊆ init(); (open(); use(*; close())*
server[ ... ] :: pool : PoolType ▸ server : Srvtype
Sharing and Resource Control (Example)

List $\triangleleft$ add(Elt)*
BagType $\triangleleft$ !( pick()List & drop(List^0) )
r : BagType

server [
  *login() =
    let l = new ListO in
    new [
      buy(i) = l.add(i)
      quit() = r.drop(l)
    ]
  dump() = r.pick()
]
Road Map

- **Specifications**
  - Operational vs. Logical Specifications
  - Logics (review)
  - Program Logics
  - Modal Logics

- **Logics for Concurrency and Distribution**
  - Hennessy-Milner Logics
  - μ-Calculi
  - Spatial Logics

- **Verification Techniques**
  - Model Checking
  - Proof Systems
  - Type Systems

- **References and (lots of) Further Reading**
  - Check the Spatial Logic Model Cheker web site for an up to date annotated bibliography on logics for concurrency and distribution