Simple Properties of Mobile Computation

- We have been looking for ways to express properties of mobile computations, E.g.:
  - “Here today, gone tomorrow.”
  - “Eventually the agent crosses the firewall.”
  - “Every agent carries a suitcase.”
  - “Somewhere there is a virus.”
  - “There is always at most one entity called $n$ here.”

- As with properties of ordinary concurrent computations, formalization options include:
  - Type systems (limited).
  - Equational reasoning (hard).
  - Reasoning on traces (ugly).
  - Reasoning via modal/temporal logics (a popular compromise).
Harder Properties

- **Moreover, we would like to express properties of unique, private, hidden, and secret names:**
  - “The applet is placed in a private sandbox.”
  - “The key exchange happens in a secret location.”
  - “A shared private key is established between two locations.”
  - “A fresh nonce is generated and transmitted.”

- **Crucial to expressing this kind of properties is devising new logical quantifiers for fresh and hidden entities:**
  - “There is a fresh (never used before) name such that …”
  - “There is a hidden (unnamable) location such that …”
  - N.B.: standard quantifiers are problematic. “There exists a sandbox containing the applet” is rather different from “There exists a fresh sandbox containing the applet” and from “There exists a hidden sandbox containing the applet.”
Approach

• Use a specification logic grounded in an operational model of mobility. (So soundness is not an issue.)

• Find ways of expressing properties of dynamically changing structures of locations.
  • Previous work [POPL’00].

• Find ways of talking about hidden names. We split it into two logical tasks:
  • Find ways of quantifying over fresh names. We adopt a recent solution [Gabbay-Pitts].
  • Find ways of revealing hidden names, so we can talk about them.
  • Combine the two, to quantify over hidden locations.
    “There is a hidden location …” represented as:
    “There is a fresh name that can be used to reveal (mention) the hidden name of a location …”.
Spatial Logics

• We want to describe mobile behaviors. The *ambient calculus* provides an operational model, where spatial structures (agents, networks, etc.) are represented by nested locations.

• We also want to specify mobile behaviors. To this end, we devise an *ambient logic* that can talk about spatial structures.

**Processes**

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(void)</td>
</tr>
<tr>
<td>n[P]</td>
<td>(location)</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
</tr>
</tbody>
</table>

**Formulas**

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(there is nothing here)</td>
</tr>
<tr>
<td>n[A]</td>
<td>(there is one thing here)</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>

**Trees**

- (void)
- (location)
- (composition)
Spatial Structures

- Our basic model of space is going to be *finite-depth edge-labeled unordered trees* (c.f. semistructured data, XML). For short: *spatial trees*, represented by a syntax of *spatial expressions*. Unbounded resources are represented by infinite branching:

```plaintext
Cambridge
   Eagle
      chair chair glass glass glass ... 
         pint pint pint pint ... 

Cambridge[Eagle[chair[0] | chair[0] | !glass[pint[0]]]] | ...
```
Ambient Structures

• These spatial expressions/trees are a subset of ambient expressions/trees, which can represent both the spatial and the temporal aspects of mobile computation.

• An ambient tree is a spatial tree with, possibly, threads at each node that can locally change the shape of the tree.

\[ a[c[out a. in b. P]] \mid b[0] \]
Mobility

- *Mobility* is change of spatial structures over time.

\[
a[Q \mid c[\text{out } a. \text{ in } b. \ P]] \quad \mid b[R]
\]
Mobility

• *Mobility* is change of spatial structures over time.

\[ a \rightarrow c \rightarrow b \]

\[ a[Q] \quad | \quad c[\text{in } b. \ P] \quad | \quad b[R] \]
Mobility

- *Mobility* is change of spatial structures over time.

\[
a[Q] \quad | \quad b[R \mid c[P]]
\]
Properties of Mobile Computation

• These often have the form:
  • Right now, we have a spatial configuration, and later, we have another spatial configuration.
  • E.g.: Right now, the agent is outside the firewall, …

![Diagram showing agent and firewall]

Now
Properties of Mobile Computation

- These often have the form:
  - Right now, we have a spatial configuration, and later, we have another spatial configuration.
  - E.g.: Right now, the agent is outside the firewall, and later (after running an authentication protocol), the agent is inside the firewall.
Ambient Calculus

\[ P \in \Pi ::= \text{Processes} \]

\[ M ::= \text{Messages} \]

\( (\forall n)P \) restriction

\( 0 \) inactivity

\( P \parallel P' \) parallel

\( M[P] \) ambient

\( !P \) replication

\( M.P \) exercise a capability

\( (n).P \) input locally, bind to \( n \)

\( \langle M \rangle \) output locally (async)

\( n[] \triangleq n[0] \)

\( M \triangleq M.0 \) (where appropriate)
Reduction Semantics

- **A structural congruence relation** $P \equiv Q$:
  - On spatial expressions, $P \equiv Q$ iff $P$ and $Q$ denote the same tree. So, the syntax modulo $\equiv$ is a notation for spatial trees.
  - On full ambient expressions, $P \equiv Q$ if in addition the respective threads are “trivially equivalent”.
  - Prominent in the definition of the logic.

- **A reduction relation** $P \rightarrow^* Q$:
  - Defining the meaning of mobility and communication actions.
  - Closed up to structural congruence:
    $$P \equiv P', \ P' \rightarrow^* Q', \ Q' \equiv Q \implies P \rightarrow^* Q$$
Restriction (much as in the $\pi$-calculus)

- $(\forall n)P$
  - “The name $n$ is known only inside $P$.”
  - “Create a new name $n$ and use it in $P$.”
  - It extrudes (floats) because it represents knowledge, not behavior:

$$\begin{align*}
(\forall n)P & \equiv (\forall m)(P[n\leftarrow m]) \\
(\forall n)0 & \equiv 0 \\
(\forall n)(\forall m)P & \equiv (\forall m)(\forall n)P \\
(\forall n)(P|Q) & \equiv P|(\forall n)Q \quad \text{if } n \notin \text{fn}(P) \\
(\forall n)(m[P]) & \equiv m[(\forall n)P] \quad \text{if } n \neq m
\end{align*}$$

- Uses or restriction:
  - Initially to represent private channels.
  - Later, to represent private names of any kind:
    - Channels, Locations, Nonces, Cryptokeys, …
Modal Logics

• In a modal logic, the truth of a formula is relative to a state (called a world).
  • Temporal logic: current time.
  • Program logic: current store contents.
  • Epistemic logic: current knowledge. Etc.

• In our case, the truth of a space-time modal formula is relative to the here and now of a process.
  • The formula $n[0]$ is read:
    there is here and now an empty location called $n$
  • The operator $n[\ Diamond]$ is a single step in space (akin to the temporal next), which allows us talk about that place one step down into $n$.
  • Other modal operators talk about undetermined times (in the future) and undetermined places (in the location tree).
Logical Formulas

\( \mathcal{A} \in \Phi ::= \) Formulas

- \( T \) true
- \( \neg \mathcal{A} \) negation
- \( \mathcal{A} \lor \mathcal{A}' \) disjunction
- \( \mathcal{0} \) void
- \( \eta[\mathcal{A}] \) location
- \( \mathcal{A}@\eta \) location adjunct
- \( \mathcal{A} | \mathcal{A}' \) composition
- \( \mathcal{A} @ \mathcal{A}' \) composition adjunct
- \( \eta \mathcal{A} \) revelation
- \( \mathcal{A} \Theta \eta \) revelation adjunct
- \( \Diamond \mathcal{A} \) somewhere modality
- \( \Box \mathcal{A} \) sometime modality
- \( \forall x. \mathcal{A} \) universal quantification over names

(\( \eta \) is a name \( n \) or a variable \( x \))
Simple Examples

1: \( p[T] \mid T \)
   there is a location \( p \) here (and possibly something else)

2: \( \lozenge 1 \)
   somewhere there is a location \( p \)

3: \( 2 \Rightarrow \Box 2 \)
   if there is a \( p \) somewhere, then forever there is a \( p \) somewhere

4: \( p[q[T] \mid T] \mid T \)
   there is a \( p \) with a child \( q \) here

5: \( \lozenge 4 \)
   somewhere there is a \( p \) with a child \( q \)
### Satisfaction Relation

- $P \models T$
- $P \models \neg \mathcal{A}$ \iff $\neg P \models \mathcal{A}$
- $P \models \mathcal{A} \lor B$ \iff $P \models \mathcal{A} \lor P \models B$
- $P \models 0$ \iff $P \equiv 0$
- $P \models n[\mathcal{A}]$ \iff $\exists P'. P \equiv n[P'] \land P' \models \mathcal{A}$
- $P \models \mathcal{A}@n$ \iff $n[P] \models \mathcal{A}$
- $P \models \mathcal{A} \mid B$ \iff $\exists P', P''. P \equiv P' \mid P'' \land P' \models \mathcal{A} \land P'' \models B$
- $P \models \mathcal{A} \triangleright B$ \iff $\forall P'. P' \models \mathcal{A} \Rightarrow P \mid P' \models B$
- $P \models n@\mathcal{A}$ \iff $\exists P'. P \equiv (\forall n)P' \land P' \models \mathcal{A}$
- $P \models \mathcal{A} \trianglerighteq n$ \iff $(\forall n)P \models \mathcal{A}$
- $P \models \mathcal{A} \diamondsuit$ \iff $\exists P'. P \downarrow \ast P' \land P' \models \mathcal{A}$
- $P \models \mathcal{A} \lozenge$ \iff $\exists P'. P \rightarrow \ast P' \land P' \models \mathcal{A}$
- $P \models \forall x. \mathcal{A}$ \iff $\forall m: \Lambda. P \models \mathcal{A}\{x \leftarrow m\}$

$P \downarrow P'$ \iff $\exists n, P''. P \equiv n[P'] \mid P''$; $\downarrow^*$ is the refl-trans closure of $\downarrow$.
**Basic Fact**

- **Satisfaction is invariant under structural congruence:**
  \[ P \models \mathcal{A}, \; P \equiv P' \; \Rightarrow \; P' \models \mathcal{A} \]
  I.e.: \( \{ P \in \Pi \mid P \models \mathcal{A} \} \) is closed under \( \equiv \).

- **Hence, formulas describe congruence-invariant properties.**
  - In particular, formulas describe properties of spatial trees.
  - N.B.: Most process logics describe bisimulation-invariant properties.
Basic Tree Formulas

\[ P \vdash 0 \quad \triangleq \quad P \equiv 0 \]
\[ P \vdash n[A] \quad \triangleq \quad \exists P' \in \Pi. \ P \equiv n[P'] \land P' \vdash A \]
\[ P \vdash A \mid B \quad \triangleq \quad \exists P', P'' \in \Pi. \ P \equiv P' \mid P'' \land P' \vdash A \land P'' \vdash B \]
\[ P \vdash A @ n \quad \triangleq \quad n[P] \vdash A \]
\[ P \vdash A \triangleright B \quad \triangleq \quad \forall P' \in \Pi. \ P' \vdash A \Rightarrow P \mid P' \vdash B \]

- \[ 0 \] : there is no structure here now.
- \[ n[A] \] : there is a location \[ n \] with contents satisfying \[ A \].
- \[ A \mid B \] : there are two structures satisfying \[ A \] and \[ B \].
- \[ A @ n \] : when the current structure is placed in a location \[ n \], the resulting structure satisfies \[ A \].
- \[ A \triangleright B \] : when the current structure is composed with one satisfying \[ A \], the resulting structures satisfies \[ B \].
Satisfaction for Basic Trees

- $\models 0$

- $\models n[A]$ if $P \models A$

- $\models A \mid B$ if $P \models A$ and $Q \models B$

- $\models A \triangleright n$ if $P \models A$

- $\models A \triangleright B$ if for all $Q \models A$ we have $P \triangleright Q \models B$
Satisfaction for Somewhere/Sometime

\[ P \models \Diamond A \quad \text{if} \quad Q \models A \]

\[ P \models \Diamond A \quad \text{if} \quad P \rightarrow^* Q \quad \text{and} \quad Q \models A \]
Satisfaction for Revelation

- Trees with hidden labels:

\[
P \overset{m}{\rightarrow} P \overset{n}{\leftarrow} P^\{m\leftarrow n\} = P^\{m\leftarrow n\}
\]

\[
P \overset{n}{\rightarrow} P \overset{n}{\leftarrow} P\overset{n\overset{\models}{\rightarrow} A_n}{\Rightarrow} \text{ if } P \overset{n}{\leftarrow} A \overset{n}{\rightarrow}
\]

\[
P \overset{n}{\rightarrow} P \overset{n}{\leftarrow} P\overset{A\overset{n}{\rightarrow}}{\Rightarrow} \text{ if } P \overset{n}{\leftarrow} A
\]
Revelation

\[ P \models n \circ \overline{A} \triangleq \exists P' \in \Pi. P \equiv (\forall n)P' \land P' \models \overline{A} \]

- **\( n \circ \overline{A} \)** is read, informally:
  - **Reveal** a private name as \( n \) and check that the revealed process satisfies \( \overline{A} \).
  - Pull out (by extrusion) a \( (\forall n) \) binder, and check that the process stripped of the binder satisfies \( \overline{A} \).

- **Examples:**
  - \( n \circ n[0] \): reveal a restricted name (say, \( p \)) as \( n \) and check the presence of an empty \( n \) location in the revealed process.
    \[
    (\forall p)p[0] \models n \circ n[0]
    \]
    because \( (\forall p)p[0] \equiv (\forall n)n[0] \) and \( n[0] \models n[0] \)
• $0 \models n\nabla 0$ because $0 \equiv (\forall n)0$ and $0 \models 0$
• $m[0] \models n\nabla T$ because $m[0] \equiv (\forall n)m[0]$ and $m[0] \models T$
• $n[0] \not\equiv n\nabla T$ because $n[0] \not\equiv (\forall n)...$

• Therefore, the set of processes satisfying $n\nabla A$ is:
  • closed under $\alpha$-variants
  • closed under $\equiv$-variants
  • not closed under changes in the set of free names
  • not closed under reduction (free names may disappear)
  • not closed under any equivalence that includes reduction
  • still ok for temporal reasoning: $\neg n\nabla A \land \Diamond n\nabla A$
## Derived Formulas

<table>
<thead>
<tr>
<th>Formula</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Box A )</td>
<td>( \neg \Diamond \neg A )</td>
</tr>
<tr>
<td>( \neg \Box A )</td>
<td>( \neg \neg \neg A )</td>
</tr>
<tr>
<td>( \neg \neg \neg A )</td>
<td>( \neg \Diamond \neg A )</td>
</tr>
<tr>
<td>( \neg A \rightarrow F )</td>
<td>( \neg A \rightarrow F )</td>
</tr>
<tr>
<td>( A \rightarrow F )</td>
<td>( A \rightarrow F )</td>
</tr>
<tr>
<td>( \neg A \rightarrow F )</td>
<td>( \neg A \rightarrow F )</td>
</tr>
</tbody>
</table>

\( P \models - \iff P \models A \rightarrow P \models B \)

\( P \models - \iff P \models A \land P \models B \)

\( P \models - \iff \exists m \in A. P \models A[x \leftarrow m] \)

\( P \models - \iff \forall P' \in \Pi. P \downarrow^* P' \implies P' \models A \)

\( P \models - \iff \forall P' \in \Pi. P \rightarrow^* P' \implies P' \models A \)

\( P \models - \iff \forall P' \in \Pi. P \models A \rightarrow P' \models F \)

\( P \models - \iff \forall P' \in \Pi. \neg P' \models A \)

\( \exists P' \in \Pi. P' \models A \)
### Derived Formulas: Revelation

<table>
<thead>
<tr>
<th>Condition</th>
<th>Definition</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \diamond n )</td>
<td>( \equiv \neg n \circledast T )</td>
<td>( P \models - \iff \neg \exists P' \in \Pi. P \equiv (\forall n)P' ) ( \iff n \in \text{fn}(P) )</td>
</tr>
<tr>
<td>closed</td>
<td>( \equiv \neg \exists x. \Diamond x )</td>
<td>( P \models - \iff \neg \exists n \in \Lambda. n \in \text{fn}(P) )</td>
</tr>
<tr>
<td>separate</td>
<td>( \equiv \neg \exists x. \Diamond x \cup \Diamond x )</td>
<td>( P \models - \iff \neg \exists n \in \Lambda. P' \in \Pi, P'' \in \Pi. P \equiv P' \cup P'' \land n \in \text{fn}(P') \land n \in \text{fn}(P'') )</td>
</tr>
</tbody>
</table>

**Examples:**

- \( n[] \models \diamond n \)
- \( (\forall p)p[] \models \text{closed} \)
- \( n[] \cup m[] \models \text{separate} \)
From Satisfaction to (Propositional) Logic

• Propositional validity

\[ \text{\textit{vld}} \mathcal{A} \triangleq \forall P \in \Pi. P \vdash \mathcal{A} \quad \mathcal{A} \text{ (closed) is valid} \]

• Sequents

\[ \mathcal{A} \vdash B \triangleq \forall P \in \Pi. P \vdash \mathcal{A} \Rightarrow P \vdash B \]

• Rules

\[ \mathcal{A}_1 \vdash B_1; \ldots; \mathcal{A}_n \vdash B_n \ \{ \mathcal{A} \vdash B \triangleq (n \geq 0) \]
\[ \mathcal{A}_1 \vdash B_1 \land \ldots \land \mathcal{A}_n \vdash B_n \Rightarrow \mathcal{A} \vdash B \]

(N.B.: all the rules shown later are validated accordingly.)

• Conventions:

- \( \vdash \) means \( \vdash \) in both directions

- \{ \} means \{ \} in both directions
• Logical axioms and rules.
  • Rules of propositional logic (standard).
  • Rules of location and composition
    \[ \mathcal{A} \vdash C \rightarrow B \quad \Rightarrow \quad \mathcal{A} \vdash C \triangleright B \quad \triangleright \quad \text{adjunction} \]
  • Rules of revelation
    \[ \eta \triangleright \mathcal{A} \vdash B \quad \Rightarrow \quad \mathcal{A} \vdash B \triangleright \eta \quad \triangleright \quad \text{adjunction} \]
    \[ \{ (\neg \mathcal{A}) \triangleright x \vdash \neg (\mathcal{A} \triangleright x) \} \quad \triangleright \quad \text{is self-dual} \]
  • Rules of ♠ and ◇ modalities (standard S4, plus some)
  • Rules of quantification (standard, but for name quantifiers)
  • A large collection of logical consequences.
Ex: Immovable Object vs. Irresistible Force

\[
\begin{align*}
Im & \equiv T \triangleright \Box (obj[] \mid T) \\
Ir & \equiv T \triangleright \Box \Diamond \neg (obj[] \mid T)
\end{align*}
\]

\[
\begin{align*}
Im \mid Ir & \vdash (T \triangleright \Box (obj[] \mid T)) \mid T \\
& \vdash \Box (obj[] \mid T) \\
& \vdash \Diamond \Box (obj[] \mid T)
\end{align*}
\]

\[
\begin{align*}
Im \mid Ir & \vdash T \mid (T \triangleright \Box \Diamond \neg (obj[] \mid T)) \\
& \vdash \Box \Diamond \neg (obj[] \mid T) \\
& \vdash \neg \Diamond \Box (obj[] \mid T)
\end{align*}
\]

Hence: \( Im \mid Ir \vdash F \)

\[
\begin{align*}
A \vdash T \\
(A \triangleright B) \mid A \vdash B \\
A \vdash \Diamond A \\
A \vdash T \\
\Diamond \neg A \vdash \neg \Diamond A \\
\neg A \vdash \neg \Diamond A \\
A \land \neg A \vdash F
\end{align*}
\]
Example: Thief!

- A shopper is likely to pull out a wallet. A thief is likely to grab it.

\[
\text{Shopper} \triangleq \\
\text{Person}\left[\text{Wallet}[$] \mid \text{T}\right] \land \\
\lozenge\left(\text{Person}\left[\text{NoWallet} \mid \text{Wallet}[$]\right]\right)
\]

\[
\text{NoWallet} \triangleq \neg\left(\text{Wallet}[$] \mid \text{T}\right)
\]

\[
\text{Thief} \triangleq \text{Wallet}[$] \triangleright \lozenge\text{NoWallet}
\]

- By simple logical deductions involving laws of \(\triangleright\) and \(\lozenge\):

\[
\text{Shopper} \mid \text{Thief} \implies \\
\left(\text{Person}\left[\text{Wallet}[$] \mid \text{T}\right] \mid \text{Thief}\right) \land \\
\lozenge\left(\text{Person}\left[\text{NoWallet} \mid \text{NoWallet}\right]\right)
\]
Fresh-Name Quantifier

\[ P \models \forall x. \mathcal{A} \iff \exists m \in \Lambda. m \notin fn(P, \mathcal{A}) \land P \models \mathcal{A}[x \leftarrow m] \]

- **C.f.:** \( P \models \exists x. \mathcal{A} \iff \exists m \in \Lambda. P \models \mathcal{A}[x \leftarrow m] \)
- **Actually definable (metatheoretically, as an abbreviation):**
  \[ \forall x. \mathcal{A} \iff \exists x. x \#(fn(\mathcal{A}) - \{x\}) \land x \otimes T \land \mathcal{A} \]

- **Fundamental “freshness” property (Gabbay-Pitts):**
  \[ \forall x. \mathcal{A} \iff \exists m \in \Lambda. m \notin fn(P, \mathcal{A}) \land P \models \mathcal{A}[x \leftarrow m] \]
  \[ \iff \forall m \in \Lambda. m \notin fn(P, \mathcal{A}) \Rightarrow P \models \mathcal{A}[x \leftarrow m] \]

  because *any fresh name as as good as any other.*

- **Very nice properties:**
  - \( \forall x. \mathcal{A} \Rightarrow \forall x. \mathcal{A} \Rightarrow \exists x. \mathcal{A} \)
  - \( \neg \forall x. \mathcal{A} \iff \forall x. \neg \mathcal{A} \)
  - \( \forall x. (\mathcal{A} \mid \mathcal{B}) \iff (\forall x. \mathcal{A}) \mid (\forall x. \mathcal{B}) \)
  - \( \Diamond \forall x. \mathcal{A} \iff \forall x. \Diamond \mathcal{A} \)
### Hidden-Name Quantifier

$$(\forall x)A \iff \forall x.x@A$$

- **Example:** $$(\forall x)x[T] = \forall x.x@x[T]$$
  - “for hidden $$x$$, we find a location called $$x$$” = “for fresh $$x$$, we reveal a hidden name as $$x$$, then we find a location called $$x$$”
  - $$(\forall n)n[] \models (\forall x)x[T]$$ because $$(\forall n)n[] \models \forall x.x@x[T]$$ because $$(\forall n)n[] \models n@n[T]$$ (where $$n \notin fn((\forall n)n[])$$).

- **Other examples**
  - $$(\forall m)m[] \models (\forall x)n[]$$
  - $$(\forall n)n[] \models (\forall n)n[] \not\models (\forall x)(x[] \mid x[])$$
  - $$(\forall n)(n[] \mid n[]) \not\models (\forall x)x[] \mid (\forall x)x[]$$
A Good Property

- A property not shared by other candidate definitions (it is even derivable within the logic):

\[(\forall x)(\mathcal{A}\{n \leftarrow x\}) \land n \circ T \vdash n \circ \mathcal{A} \quad \text{where} \quad x \notin \text{fv}(\mathcal{A})\]

It implies:

\[P \models \mathcal{A} \Rightarrow (\forall n)P \models (\forall x)(\mathcal{A}\{n \leftarrow x\})\]

\[P \models n \circ \mathcal{A} \Rightarrow P \models (\forall x)(\mathcal{A}\{n \leftarrow x\})\]

\[P \models (\forall x)(\mathcal{A}\{n \leftarrow x\}) \land n \notin \text{fn}(P) \Rightarrow P \models n \circ \mathcal{A}\]
**Example: Key Sharing**

- Consider a situation where “a hidden name \( x \) is shared by two locations \( n \) and \( m \), and is not known outside those locations”.

\[
(\forall x) \ (n[\odot x] \cup m[\odot x])
\]

- \( P \models (\forall x) \ (n[\odot x] \cup m[\odot x]) \)

\[
\iff \exists r \in \Lambda. \ r \notin \text{fn}(P) \cup \{n,m\} \land \exists R', R'' \in \Pi. \ P \equiv (\forall r)(n[R'] \cup m[R'']) \land r \in \text{fn}(R') \land r \in \text{fn}(R'')
\]

- E.g.: take \( P = (\forall p) \ (n[p][] \cup m[p][]). \)

- A protocol establishing a shared key should satisfy:

\[
\Diamond (\forall x) \ (n[\odot x] \cup m[\odot x])
\]
Applications

- Verifying security+mobility protocols.
- Modelchecking security+mobility assertions:
  - If $P$ is $\lnot$-free and $\mathcal{A}$ is $\triangleright$-free, then $P \models \mathcal{A}$ is decidable.
  - This provides a way of mechanically checking (certain) assertions about (certain) mobile processes.
- Expressing mobility/security policies of host sites.
  (Conferring more flexibility than just sandboxing the agent.)
- Just-in-time verification of code containing mobility instructions (by either modelchecking or proof-carrying code).
Conclusions

• The novel aspects of our logic lie in its explicit treatment of space and of the evolution of space over time (mobility). The logic has a linear flavor in the sense that space cannot be instantly created or deleted, although it can be transformed over time.

• These ideas can be applied to any process calculus that embodies a distinction between spatial and temporal operators.

• Our logical rules arise from a particular model. This approach makes the logic very concrete (and sound), but raises questions of logical completeness, which are being investigated.