A Decade of Dependent Session Types

Bernardo Toninho (NOVA)
Luís Caires (NOVA) and Frank Pfenning (CMU)

September 6, 2021

Outline

Session Types

A bit of history

- Session types were developed in the 90s [Honda93,HVK98].
- Originally a typing system for a π-calculus.
- Structure communication around the concept of a **session**.

Session Types

A bit of history

- Session types were developed in the 90s [Honda93,HVK98].
- Originally a typing system for a π-calculus.
- Structure communication around the concept of a **session**.

Session

Predetermined sequence of interactions along a (session) channel:

- "Input a number, output a string and terminate."
- "Either output or input a number."

Session Types

A bit of history

- Session types were developed in the 90s [Honda93,HVK98].
- Originally a typing system for a π-calculus.
- Structure communication around the concept of a **session**.

Session

Predetermined sequence of interactions along a (session) channel:

- "Input a number, output a string and terminate."
- "Either output or input a number."

Session $\approx$ Communication Protocol

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

- "Input a number, output a string and terminate." $\approx$ Int $\multimap$ String $\otimes$ **1** $\quad (T_1)$

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

- "Input a number, output a string and terminate." $\approx$ Int $\multimap$ String $\otimes$ **1** $\quad (T_1)$

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

- "Input a number, output a string and terminate." $\approx$ Int $\multimap$ String $\otimes$ **1** $\quad (T_1)$

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

- "Input a number, output a string and terminate." $\approx \mathsf{Int} \multimap \mathsf{String} \otimes \mathbf{1} \quad (T_1)$

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

- "Input a number, output a string and terminate." $\approx$ Int $\multimap$ String $\otimes$ **1** $\quad (T_1)$
- "Send a number, input a string and terminate." $\approx$ Int $\otimes$ String $\multimap$ **1** $\quad (T_2)$

Session Types

Types as Protocols

- Session types **are** descriptions of comm. behavior, assigned to **channels**.
- A way of guaranteeing communication discipline, **statically**.
- Intrinsic notion of duality: Send/Receive, Offer choice/Select.
- Duality ensures session fidelity (and deadlock-freedom, with some caveats).

Session Types

- "Input a number, output a string and terminate." $\approx$ Int $\multimap$ String $\otimes$ **1** $\quad (T_1)$
- "Send a number, input a string and terminate." $\approx$ Int $\otimes$ String $\multimap$ **1** $\quad (T_2)$

$$c : T_1 \vdash P \qquad c : T_2 \vdash Q$$

- T_1 and T_2 are **dual** ($\overline{T_1} = T_2$), no communication errors between P and Q!

Session Types
Properties

Session types statically guarantee:

- Session fidelity (“No communication errors.”)
- Deadlock-freedom…

Session Types

Properties

Session types statically guarantee:

- Session fidelity (“No communication errors.”)
- Deadlock-freedom…
  - If threads communicate on exactly **one** session channel
  - Excludes session interleaving
  - Excludes higher-order sessions (sending channels over channels)

Session Types
Properties

Session types statically guarantee:

- Session fidelity (“No communication errors.”)
- Deadlock-freedom…
  - If threads communicate on exactly **one** session channel
  - Excludes session interleaving
  - Excludes higher-order sessions (sending channels over channels)

Progress with session interleaving [DLY07] via sophisticated machinery.

Session Types

Limitations

- Deadlock-freedom only in (very) restricted settings.
- Session typing only really about **two** communicating peers.

Session Types

Limitations

- Deadlock-freedom only in (very) restricted settings.
- Session typing only really about **two** communicating peers.
- Express only fairly basic protocols (e.g., send/receive, choice/select).

Session Types

Limitations

- Deadlock-freedom only in (very) restricted settings.
- Session typing only really about **two** communicating peers.
- Express only fairly basic protocols (e.g., send/receive, choice/select).
- Sometimes, simple i.o. communication behavior is not enough!
  - “balance inquiry for authenticated user receives a signed statement”
  - “ATM deducts a fee of at most $2 per transaction”
  - …

Session Types

Addressing the Limitations

Multiparty Session Types [HYC08]

- Types can specify interactions between more than two peers.
- Deadlock-freedom in (well-formed) multiparty sessions.
- More complex system (global types, local types, projection, etc.)

Session Types

Addressing the Limitations

Multiparty Session Types [HYC08]

- Types can specify interactions between more than two peers.
- Deadlock-freedom in (well-formed) multiparty sessions.
- More complex system (global types, local types, projection, etc.)

Dependent Session Types [TCP11]

- Beyond simple protocols as types.
- Types can express **arbitrary** properties of exchanged data.
- Based on a computational interpretation of linear logic.

Logical Session Types

Propositions as Sessions

Session Types and Propositional Linear Logic [CP10]

- Its possible to interpret session types as linear logic propositions.
- Linear logic proofs as (process) typing derivations.
- Proof simplification as communication.

Logical Session Types

Propositions as Sessions

Session Types and Propositional Linear Logic [CP10]

- Its possible to interpret session types as linear logic propositions.
- Linear logic proofs as (process) typing derivations.
- Proof simplification as communication.

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.

Logical Session Types

Propositions as Sessions

Session Types and Propositional Linear Logic [CP10]

- Its possible to interpret session types as linear logic propositions.
- Linear logic proofs as (process) typing derivations.
- Proof simplification as communication.

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.
- Multiplicative Conjunction ($A \otimes B$): Send a channel of type A and cont. as B.

Logical Session Types

Propositions as Sessions

Session Types and Propositional Linear Logic [CP10]

- Its possible to interpret session types as linear logic propositions.
- Linear logic proofs as (process) typing derivations.
- Proof simplification as communication.

- Linear Implication ($A \multimap B$): Receive a channel of type A and continue with B.
- Multiplicative Conjunction ($A \otimes B$): Send a channel of type A and cont. as B.
- Additive Conjunction ($A \,\&\, B$): Receive either inl and continue as A or inr and continue as B.
- Additive Disjunction ($A \oplus B$): Send inl and continue as A or inr and cont. as B.

Logical Session Types

Propositions as Sessions

- Proof composition (cut) as process composition.
- Global progress “for free” (with interleaved and higher-order sessions).
- Termination, cut-elimination, confluence.
- A unifying framework to explore various extensions of session types:
  - Classical linear logic [W12,CPT16]
  - Dependent session types [TCP11,PCT11,TY18]
  - Structural recursion for session types [TCP14,LM16]
  - Sharing in sessions [ALM16,BP17,RC21]
  - ...

Outline

Before our Work

Our Work

(Some of) what came after

Open Problems and Ongoing Work

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **Propositional** linear logic as session types:
  - Input and output of session channels ($A \multimap B$ and $A \otimes B$)
  - Choice and selection of alternatives ($A \mathbin{\&} B$ and $A \oplus B$)
  - Replicated servers ($!A$)
  - Termination or inaction ($\mathbf{1}$)

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **Propositional** linear logic as session types:
  - Input and output of session channels ($A \multimap B$ and $A \otimes B$)
  - Choice and selection of alternatives ($A \& B$ and $A \oplus B$)
  - Replicated servers ($!A$)
  - Termination or inaction ($\mathbf{1}$)

Types express very limited protocols...

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
  - Values in domain of quantification (τ) from a dependent type theory.
  - $\$\tau \multimap A$ as shorthand for $\forall y{:}\tau.A$ if y not free in A
  - $\$\tau \otimes A$ as shorthand for $\exists y{:}\tau.A$ if y not free in A
- Types can now express **contracts** on communicated data.

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
  - Values in domain of quantification (τ) from a dependent type theory.
  - $\$\tau \multimap A$ as shorthand for $\forall y{:}\tau.A$ if y not free in A
  - $\$\tau \otimes A$ as shorthand for $\exists y{:}\tau.A$ if y not free in A
- Types can now express **contracts** on communicated data.

$$P :: z : \$\mathsf{nat} \multimap \$\mathsf{nat} \otimes \mathbf{1}$$

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
  - Values in domain of quantification (τ) from a dependent type theory.
  - $\$\tau \multimap A$ as shorthand for $\forall y{:}\tau.A$ if y not free in A
  - $\$\tau \otimes A$ as shorthand for $\exists y{:}\tau.A$ if y not free in A
- Types can now express **contracts** on communicated data.

$$P :: z : \forall n{:}\mathsf{nat}.\, \exists n'{:}\mathsf{nat}.\, \$(n' = n + 1) \otimes \mathbf{1}$$

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
  - Values in domain of quantification (τ) from a dependent type theory.
  - $\$\tau \multimap A$ as shorthand for $\forall y{:}\tau.A$ if y not free in A
  - $\$\tau \otimes A$ as shorthand for $\exists y{:}\tau.A$ if y not free in A
- Types can now express **contracts** on communicated data.

$$P :: z : \forall n{:}\mathsf{nat}.\, \exists n'{:}\mathsf{nat}.\, \$(n' = n + 1) \otimes \mathbf{1}$$

Dependent Session Types

First-Order Propositions as Dependent Sessions

- **First-order** linear logic as session types:
  - Universal Quantification ($\forall x{:}\tau.A$): Receive $M{:}\tau$ and continue as $A\{M/x\}$.
  - Existential Quantification ($\exists x{:}\tau.A$): Send $M{:}\tau$ and continue as $A\{M/x\}$.
  - Values in domain of quantification (τ) from a dependent type theory.
  - $\$\tau \multimap A$ as shorthand for $\forall y{:}\tau.A$ if y not free in A
  - $\$\tau \otimes A$ as shorthand for $\exists y{:}\tau.A$ if y not free in A
- Types can now express **contracts** on communicated data.

$$P :: z : \forall n{:}\mathsf{nat}.\, \exists n'{:}\mathsf{nat}.\, \$(n' = n + 1) \otimes \mathbf{1}$$

Processes send and receive **proof objects** that witness the desired properties.

Dependent Session Types

Examples with Proof-Carrying

- PDF indexing service

$$\mathsf{index}_1 \quad : \quad !(\mathsf{file} \multimap \mathsf{file} \otimes \mathbf{1})$$

Dependent Session Types

Examples with Proof-Carrying

- PDF indexing service

$$\begin{array}{lcl} \mathsf{index}_1 & : & !(\mathsf{file} \multimap \mathsf{file} \otimes \mathbf{1}) \\ \mathsf{index}_2 & : & !(\forall f{:}\mathsf{file}.\, \mathsf{ispdf}(f) \multimap \exists g{:}\mathsf{file}.\, \mathsf{ispdf}(g) \otimes \mathbf{1}) \end{array}$$

Persistently offer to input a file f, **a proof that f is in PDF format**, then output a PDF file g, and **a proof that g is in PDF format** and terminate the session.

Dependent Session Types

Examples with Proof-Carrying

- PDF indexing service

$$\begin{array}{lcl} \mathsf{index}_1 & : & !(\mathsf{file} \multimap \mathsf{file} \otimes \mathbf{1}) \\ \mathsf{index}_2 & : & !(\forall f{:}\mathsf{file}.\, \mathsf{ispdf}(f) \multimap \exists g{:}\mathsf{file}.\, \mathsf{ispdf}(g) \otimes \mathbf{1}) \end{array}$$

  Persistently offer to input a file f, **a proof that f is in PDF format**, then output a PDF file g, and **a proof that g is in PDF format** and terminate the session.

- Persistent file storage

$$\mathsf{store}_1 \quad : \quad !(\mathsf{file} \multimap !(\mathsf{file} \otimes \mathbf{1}))$$

Dependent Session Types

Examples with Proof-Carrying

- PDF indexing service

$$\begin{array}{lcl} \text{index}_1 & : & !(\text{file} \multimap \text{file} \otimes \mathbf{1}) \\ \text{index}_2 & : & !(\forall f{:}\text{file}.\ \text{ispdf}(f) \multimap \exists g{:}\text{file}.\ \text{ispdf}(g) \otimes \mathbf{1}) \end{array}$$

Persistently offer to input a file f, **a proof that f is in PDF format**, then output a PDF file g, and **a proof that g is in PDF format** and terminate the session.

- Persistent file storage

$$\begin{array}{lcl} \text{store}_1 & : & !(\text{file} \multimap !(\text{file} \otimes \mathbf{1})) \\ \text{store}_2 & : & !(\forall f{:}\text{file}.\ !\exists g{:}\text{file}.\ g \doteq f \otimes \mathbf{1}) \end{array}$$

Persistently offer to input a file, then output a persistent channel for retrieving this file and a **proof that the two are equal**.

Dependent Session Types

Modalities – Proof Irrelevance

- In many examples, we want to know that proofs exist, but we do not want to transmit them

Dependent Session Types

Modalities – Proof Irrelevance

- In many examples, we want to know that proofs exist, but we do not want to transmit them
  - We can easily check ispdf(g) when using the indexing service
  - The proof of $g \doteq f$ (by reflexivity) would not be informative

Dependent Session Types

Modalities – Proof Irrelevance

- In many examples, we want to know that proofs exist, but we do not want to transmit them
  - We can easily check ispdf(g) when using the indexing service
  - The proof of $g \doteq f$ (by reflexivity) would not be informative
- Use **proof irrelevance** in type theory
- $M : [\tau]$ – M is a term of type τ that is computationally irrelevant.

Dependent Session Types

Modalities – Proof Irrelevance

- In many examples, we want to know that proofs exist, but we do not want to transmit them
  - We can easily check ispdf(g) when using the indexing service
  - The proof of $g \doteq f$ (by reflexivity) would not be informative
- Use **proof irrelevance** in type theory
- $M : [\tau]$ – M is a term of type τ that is computationally irrelevant.
- By agreement, terms $[M]$ **will be erased before transmission**.
- Typing guarantees this can be done consistently.

Dependent Session Types

Modalities – Proof Irrelevance

- Mark proofs as computationally irrelevant
- PDF indexing service

$$\mathsf{index}_2 \quad : \quad !(\forall f{:}\mathsf{file}.\, \mathsf{ispdf}(f) \multimap \exists g{:}\mathsf{file}.\, \mathsf{ispdf}(g) \otimes \mathbf{1})$$

Dependent Session Types

Modalities – Proof Irrelevance

- Mark proofs as computationally irrelevant
- PDF indexing service

$$\mathsf{index}_2 \quad : \quad !(\forall f{:}\mathsf{file}.\ \mathsf{ispdf}(f) \multimap \exists g{:}\mathsf{file}.\ \mathsf{ispdf}(g) \otimes \mathbf{1})$$

$$\mathsf{index}_3 \quad : \quad !(\forall f{:}\mathsf{file}.\ [\mathsf{ispdf}(f)] \multimap \exists g{:}\mathsf{file}.\ [\mathsf{ispdf}(g)] \otimes \mathbf{1})$$

Dependent Session Types

Modalities – Proof Irrelevance

- Mark proofs as computationally irrelevant
- PDF indexing service

$$\begin{aligned} \mathsf{index}_2 &: !(\forall f{:}\mathsf{file}.\ \mathsf{ispdf}(f) \multimap \exists g{:}\mathsf{file}.\ \mathsf{ispdf}(g) \otimes \mathbf{1}) \\ \mathsf{index}_3 &: !(\forall f{:}\mathsf{file}.\ [\mathsf{ispdf}(f)] \multimap \exists g{:}\mathsf{file}.\ [\mathsf{ispdf}(g)] \otimes \mathbf{1}) \end{aligned}$$

- Persistent file storage

$$\mathsf{store}_2 \;:\; !(\forall f{:}\mathsf{file}.\ !\exists g{:}\mathsf{file}.\ g \doteq f \otimes \mathbf{1})$$

Dependent Session Types

Modalities – Proof Irrelevance

- Mark proofs as computationally irrelevant
- PDF indexing service

$$\text{index}_2 \;:\; !(\forall f{:}\text{file}.\ \text{ispdf}(f) \multimap \exists g{:}\text{file}.\ \text{ispdf}(g) \otimes \mathbf{1})$$

$$\text{index}_3 \;:\; !(\forall f{:}\text{file}.\ [\text{ispdf}(f)] \multimap \exists g{:}\text{file}.\ [\text{ispdf}(g)] \otimes \mathbf{1})$$

- Persistent file storage

$$\text{store}_2 \;:\; !(\forall f{:}\text{file}.\ !\exists g{:}\text{file}.\ g \doteq f \otimes \mathbf{1})$$

$$\text{store}_3 \;:\; !(\forall f{:}\text{file}.\ !\exists g{:}\text{file}.\ [g \doteq f] \otimes \mathbf{1})$$

- After erasure, communication can be optimized further (via type isomorphism).

Dependent Session Types

Taking Stock

- A flexible and general framework of session type dependency.
- Session types enriched to certified contracts on exchanged data:

Dependent Session Types

Taking Stock

- A flexible and general framework of session type dependency.
- Session types enriched to certified contracts on exchanged data:
  - Arbitrary properties of data ensured statically, witnessed by proof objects.
  - Proof communication can be selectively omitted (c.f. type refinements).

Dependent Session Types

Taking Stock

- A flexible and general framework of session type dependency.
- Session types enriched to certified contracts on exchanged data:
  - Arbitrary properties of data ensured statically, witnessed by proof objects.
  - Proof communication can be selectively omitted (c.f. type refinements).
- Logical basis provides modularity.
- Type preservation and progress ensure contracts are preserved by computation/communication.

Outline

Roadmap

1. Digital signatures through modal affirmation.
2. Recursion and Sharing
3. Ergometric and Temporal Session Types
4. Richer forms of dependency

Dependent Session Types

Modalities – Affirmation [PCT11]

- In the PDF indexing example, we may want to have some evidence the two files agree.

Dependent Session Types

Modalities – Affirmation [PCT11]

- In the PDF indexing example, we may want to have some evidence the two files agree.

$$\begin{aligned}\mathsf{index}_4 \quad : \quad & !(\forall f{:}\mathsf{file}.\, [\mathsf{ispdf}(f)] \\ & \quad \multimap \exists g{:}\mathsf{file}.\, [\mathsf{ispdf}(g)] \otimes [\mathsf{agree}(g, f)] \otimes \mathbf{1})\end{aligned}$$

$\mathsf{agree}(g, f)$ if g and f differ at most in the index.

Dependent Session Types

Modalities – Affirmation [PCT11]

- In the PDF indexing example, we may want to have some evidence the two files agree.

$$\begin{aligned} \mathsf{index}_4 \quad : \quad & !(\forall f{:}\mathsf{file}.\ [\mathsf{ispdf}(f)] \\ & \multimap \exists g{:}\mathsf{file}.\ [\mathsf{ispdf}(g)] \otimes [\mathsf{agree}(g, f)] \otimes \mathbf{1}) \end{aligned}$$

  $\mathsf{agree}(g, f)$ if g and f differ at most in the index.

- Since no proof is transmitted, client may require indexer X's explicit affirmation (= digital signature)!
- Similarly, in the persistent file storage example

Dependent Session Types

Modalities – Affirmation [PCT11]

- An **affirmation** modality: “Principal K affirms property τ due to evidence M”.

Dependent Session Types

Modalities – Affirmation [PCT11]

- An **affirmation** modality: “Principal K affirms property τ due to evidence M”.
- Embodied in type $\Diamond_K \tau$:
  - A value of type $\Diamond_K \tau$ denotes a term M of type τ, **digitally signed by** K.
  - Assume some public key infrastructure.

Dependent Session Types

Modalities – Affirmation [PCT11]

- An **affirmation** modality: “Principal K affirms property τ due to evidence M”.
- Embodied in type $\Diamond_K \tau$:
  - A value of type $\Diamond_K \tau$ denotes a term M of type τ, **digitally signed by** K.
  - Assume some public key infrastructure.
  - $\Diamond_K$ is a K-indexed family of strong monads.
  - In general cannot get a value of type τ from $\Diamond_K \tau$.

Dependent Session Types

Affirmation – Example

- PDF indexing service, with indexer X

$$\mathsf{index}_5 : !(\forall f{:}\mathsf{file}.\ [\mathsf{ispdf}(f)] \multimap \exists g{:}\mathsf{file}.\ [\mathsf{ispdf}(g)] \otimes \Diamond_X [\mathsf{agree}(g, f)] \otimes \mathbf{1})$$

Dependent Session Types

Affirmation – Example

- PDF indexing service, with indexer X

$$\mathsf{index}_5 : !(\forall f{:}\mathsf{file}.\ [\mathsf{ispdf}(f)] \multimap \exists g{:}\mathsf{file}.\ [\mathsf{ispdf}(g)] \otimes \Diamond_X[\mathsf{agree}(g, f)] \otimes \mathbf{1})$$

- Persistent file storage, with file system Y

$$\mathsf{store}_4 : !(\forall f{:}\mathsf{file}.\ !\exists g{:}\mathsf{file}.\ \Diamond_Y[g \doteq f] \otimes \mathbf{1})$$

Dependent Session Types

Affirmation – Example

- PDF indexing service, with indexer X

$$\mathsf{index}_5 : !(\forall f{:}\mathsf{file}.\ [\mathsf{ispdf}(f)] \multimap \exists g{:}\mathsf{file}.\ [\mathsf{ispdf}(g)] \otimes \Diamond_X [\mathsf{agree}(g, f)] \otimes \mathbf{1})$$

- Persistent file storage, with file system Y

$$\mathsf{store}_4 : !(\forall f{:}\mathsf{file}.\ !\exists g{:}\mathsf{file}.\ \Diamond_Y [g \doteq f] \otimes \mathbf{1})$$

- Idiom $\Diamond_K[\tau]$ may transmit
  - $\langle [\,]{:}\tau \rangle_K$, a certificate, digitally signed by K affirming τ
  - Some proof that $[\tau]$ follows from affirmations by K, according to the laws of $\Diamond_K$ (e.g. K affirms that X affirms τ).

Dependent Session Types

Affirmation – Trust Axioms

- Affirmations track aspects of provenance and info. flow
  - "Diamonds are forever"
  - In general, $\not\vdash \Diamond_K \tau \rightarrow \tau$
  - Need declassification

Dependent Session Types

Affirmation – Trust Axioms

- Affirmations track aspects of provenance and info. flow
  - "Diamonds are forever"
  - In general, $\not\models \Diamond_K \tau \rightarrow \tau$
  - Need declassification
- Trust axioms
  - For specific types τ and principals K:

$$\mathsf{trust}_{K,\tau} : \Diamond_K \tau \rightarrow \tau$$

  - Implementable, in general, by stripping signature

Dependent Session Types

Affirmation – Trust Axioms

- Affirmations track aspects of provenance and info. flow
  - "Diamonds are forever"
  - In general, $\nvdash \Diamond_K \tau \to \tau$
  - Need declassification
- Trust axioms
  - For specific types τ and principals K:

$$\mathsf{trust}_{K,\tau} : \Diamond_K \tau \to \tau$$

  - Implementable, in general, by stripping signature
- Omitted proofs $[\tau]$ cannot be recovered, in general

$\nvdash [\tau] \to \tau$	not implementable, in general
$\nvdash \Diamond_K [\tau] \to \tau$	not implementable, in general

Roadmap

- Digital signatures through modal affirmation.
- Recursion and Sharing
- Ergometric and Temporal Session Types
- Richer forms of dependency

Recursion and Sharing

Motivation

- Logical types so far cannot express iterative behaviors.
- Limits applicability to many real-world examples.

Recursion and Sharing

Motivation

- Logical types so far cannot express iterative behaviors.
- Limits applicability to many real-world examples.
- Especially the case due to linearity (sessions are isolated, one-shot).
- Two approaches:

Recursion and Sharing

Motivation

- Logical types so far cannot express iterative behaviors.
- Limits applicability to many real-world examples.
- Especially the case due to linearity (sessions are isolated, one-shot).
- Two approaches:
  - Recursive and co-recursive session types [TCP13,TCP14,LM16,TY19]
  - Shared Sessions [ALM16,BP17]

Recursion and Sharing

Recursive Types and Processes

Via fixed point combinators [TCP13,TCP14]

- Ability to write recursive programs (e.g. a stream of natural numbers):

```
nats : nat -> {c:nu X.$nat * X}
nats n = {c.
  send c n
  nats (n+1)
}
```

- Combines/conflates recursion and corecursion.

Recursion and Sharing

Recursive Types and Processes

Via fixed point combinators [TCP13,TCP14]

- Ability to write recursive programs (e.g. a stream of natural numbers):

```
nats : nat -> {c:nu X.$nat * X}
nats n = {c.
  send c n
  nats (n+1)
}
```

- Combines/conflates recursion and corecursion.
- General recursion abandons logical soundness (non-termination).
- Can be recovered via syntactic means of ensuring productivity [TCP14].

Recursion and Sharing

Recursive Types and Processes

Via initial algebra and final coalgebra semantics [LM16,TY20]:

- Extend language with type functors F and their least and greatest fixed points μF and νF.
- Terms extended with appropriate operators: in, out, fold, unfold.

Recursion and Sharing

Recursive Types and Processes

Via initial algebra and final coalgebra semantics [LM16,TY20]:

- Extend language with type functors F and their least and greatest fixed points μF and νF.
- Terms extended with appropriate operators: in, out, fold, unfold.

Recursive channels for natural numbers (**finitely generated**):

$$\mathit{NC}(X) = \mathbf{1} \oplus (\mathsf{nat} \otimes X) \qquad \mathit{Nats} = \mu \mathit{NC}$$

Recursion and Sharing

Recursive Types and Processes

Via initial algebra and final coalgebra semantics [LM16,TY20]:

- Extend language with type functors F and their least and greatest fixed points μF and νF.
- Terms extended with appropriate operators: in, out, fold, unfold.

Recursive channels for natural numbers (**finitely generated**):

$$\mathit{NC}(X) = \mathbf{1} \oplus (\mathsf{nat} \otimes X) \qquad \mathit{Nats} = \mu \mathit{NC}$$

Corecursive channels for natural numbers (**finitely consumed**):

$$\mathit{NC}'(X) = (\$\mathsf{nat} \otimes X) \qquad \mathit{Nats}' = \nu \mathit{NC}'$$

Recursion and Sharing

Sharing

- Logical session types fail to capture numerous features of process calculus, even when extended recursion.
- Computation is confluent and only features “don’t care” non-determinism.
- Linearity itself can be very restrictive (e.g. well-typed compositions require sharing exactly 1 channel).
- How to recover these features? and at what cost?

Recursion and Sharing

Sharing

- Logical session types fail to capture numerous features of process calculus, even when extended recursion.
- Computation is confluent and only features “don’t care” non-determinism.
- Linearity itself can be very restrictive (e.g. well-typed compositions require sharing exactly 1 channel).
- How to recover these features? and at what cost?
  - Conflation of dual types [ALM16]
  - Manifest sharing [BP17]

Recursion and Sharing

Sharing

Conflation of dual types, in classical linear logic [ALM16]:

- $\otimes$ and $\multimap$: Sharing of multiple channels between parallel threads.
- $!$ and $?$: Access points (i.e. stateful non-determinism).

Recursion and Sharing

Sharing

Conflation of dual types, in classical linear logic [ALM16]:

- $\otimes$ and $\multimap$: Sharing of multiple channels between parallel threads.
- $\oplus$ and $\&$: Local non-determinism / failures (c.f. $P + Q$ in π-calculus).
- ! and $?$: Access points (i.e. stateful non-determinism).

Recursion and Sharing

Sharing

Conflation of dual types, in classical linear logic [ALM16]:

- $\otimes$ and $\multimap$: Sharing of multiple channels between parallel threads.
- $\oplus$ and $\&$: Local non-determinism / failures (c.f. $P + Q$ in π-calculus).
- ! and $?$: Access points (i.e. stateful non-determinism).

The price of conflation:

- $\otimes$ and $\multimap$: Deadlocks typable, termination and determinism preserved.
- $\oplus$ and $\&$: Determinism is lost.
- ! and $?$: Termination, deadlock-freedom and determinism lost.

Recursion and Sharing

Sharing

Manifest sharing [BP17]:

- Alternative interpretation of the exponential $!A$, sharing instead of copying.
- Programmatically, controlled via an acquire-release discipline.
- Manifest in the type structure via $\uparrow_{\mathsf{L}}^{\mathsf{S}} A$ and $\downarrow_{\mathsf{L}}^{\mathsf{S}} A$ (based on Benton's LNL [B94] and Reed's adjoint logic [R09]).

Recursion and Sharing

Sharing

Manifest sharing [BP17]:

- Alternative interpretation of the exponential $!A$, sharing instead of copying.
- Programmatically, controlled via an acquire-release discipline.
- Manifest in the type structure via $\uparrow_{\mathsf{L}}^{\mathsf{S}} A$ and $\downarrow_{\mathsf{L}}^{\mathsf{S}} A$ (based on Benton's LNL [B94] and Reed's adjoint logic [R09]).
- A shared session of type $\uparrow_{\mathsf{L}}^{\mathsf{S}} A$ may be acquired by clients (race for shared resource), subsequently used as A.
- A linear session of type $\downarrow_{\mathsf{L}}^{\mathsf{S}} A$ may then be released, back to shared mode.

Recursion and Sharing

Sharing

Manifest sharing [BP17]:

- Alternative interpretation of the exponential $!A$, sharing instead of copying.
- Programmatically, controlled via an acquire-release discipline.
- Manifest in the type structure via $\uparrow_{\mathsf{L}}^{\mathsf{S}} A$ and $\downarrow_{\mathsf{L}}^{\mathsf{S}} A$ (based on Benton's LNL [B94] and Reed's adjoint logic [R09]).
- A shared session of type $\uparrow_{\mathsf{L}}^{\mathsf{S}} A$ may be acquired by clients (race for shared resource), subsequently used as A.
- A linear session of type $\downarrow_{\mathsf{L}}^{\mathsf{S}} A$ may then be released, back to shared mode.
- Asynchronous π-calculus becomes encodable (non-determinism, non-termination, deadlocks).

Roadmap

- Digital signatures through modal affirmation.
- Recursion and Sharing
- Ergometric and Temporal Session Types
- Richer forms of dependency

Ergometric and Temporal Session Types

Indexed Types

- With recursive types, it becomes natural to think of indexed session types:

$$\mathsf{nats} = \$\mathsf{nat} \otimes \mathsf{nats}$$

Ergometric and Temporal Session Types

Indexed Types

- With recursive types, it becomes natural to think of indexed session types:

$$\mathsf{nats}[n] = \exists m : \mathsf{nat}.\$[m = n] \otimes \mathsf{nats}[n+1]$$

Ergometric and Temporal Session Types

Indexed Types

- With recursive types, it becomes natural to think of indexed session types:

$$\mathsf{nats}[n] = \exists m : \mathsf{nat}.\$[m = n] \otimes \mathsf{nats}[n+1]$$

$$\begin{aligned}\mathsf{queue}_A[n] = \&\{&\mathbf{ins} : A \multimap \mathsf{queue}_A[n+1],\\ &\mathbf{del} : \oplus\{\mathbf{none} : \$[n = 0] \otimes \mathbf{1},\\ &\qquad\qquad \mathbf{some} : \$[n > 0] \otimes A \otimes \mathsf{queue}_A[n-1]\}\}\end{aligned}$$

Ergometric and Temporal Session Types

Ergometric and Temporal Modalities [DHP18a,DHP18b,DP20]

- Complexity analysis of concurrent, message-passing programs.

Ergometric and Temporal Session Types

Ergometric and Temporal Modalities [DHP18a,DHP18b,DP20]

- Complexity analysis of concurrent, message-passing programs.
- Ergometric session types [DHP18a] capture exchange of potential, in the style of amortized complexity analysis ($\triangleright^r A$):

Ergometric and Temporal Session Types

Ergometric and Temporal Modalities [DHP18a,DHP18b,DP20]

- Complexity analysis of concurrent, message-passing programs.
- Ergometric session types [DHP18a] capture exchange of potential, in the style of amortized complexity analysis ($\triangleright^r A$):

$$
\begin{aligned}
\mathsf{queue}_A[n] = \&\{\mathbf{ins} : \triangleright^{2n} A \multimap \mathsf{queue}_A[n+1],\\
\mathbf{del} : \triangleright^{2} \oplus \{\mathbf{none} : \$[n = 0] \otimes \mathbf{1},\\
\mathbf{some} : \$[n > 0] \otimes A \otimes \mathsf{queue}_A[n-1]\}\}
\end{aligned}
$$

Ergometric and Temporal Session Types

Ergometric and Temporal Modalities [DHP18a,DHP18b,DP20]

- Complexity analysis of concurrent, message-passing programs.
- Ergometric session types [DHP18a] capture exchange of potential, in the style of amortized complexity analysis ($\triangleright^r A$):

$$\begin{aligned}\mathsf{queue}_A[n] = \&\{\mathbf{ins} : \triangleright^{2n} A \multimap \mathsf{queue}_A[n+1],\\ \mathbf{del} : \triangleright^{2} \oplus \{\mathbf{none} : \$[n = 0] \otimes \mathbf{1},\\ \mathbf{some} : \$[n > 0] \otimes A \otimes \mathsf{queue}_A[n-1]\}\}\end{aligned}$$

- Temporal session types [DHP18b] capture parallel complexity (span) via temporal modalities over linear time ($\circ A$, $\Box A$, $\Diamond A$).
- Can check constant number of delays between insertions and deletions in (bucket-brigade) queue.

Roadmap

- Digital signatures through modal affirmation
- Recursion and Sharing
- Ergometric and Temporal Session Types
- Richer forms of dependency

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Boolean-driven Communication

$$\texttt{if} :: \textsf{Bool} \rightarrow \textsf{stype} \rightarrow \textsf{stype} \rightarrow \textsf{stype}$$

$$\texttt{if}\ \textsf{true}\ A\ B\ =\ A \qquad \texttt{if}\ \textsf{false}\ A\ B\ =\ B$$

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B = A \qquad \texttt{if}\ \mathsf{false}\ A\ B = B$$

$$T \triangleq \forall x{:}\mathsf{bool}.\texttt{if}\ x\ (\$\mathsf{nat} \otimes \mathbf{1})\ (\$\mathsf{bool} \otimes \mathbf{1})$$

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Boolean-driven Communication

$$\mathtt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\mathtt{if}\ \mathsf{true}\ A\ B \ = \ A \qquad \mathtt{if}\ \mathsf{false}\ A\ B \ = \ B$$

$$T \triangleq \forall x{:}\mathsf{bool}.\mathtt{if}\ x\ (\$\mathsf{nat} \otimes \mathbf{1})\ (\$\mathsf{bool} \otimes \mathbf{1})$$

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B \ = \ A \qquad \texttt{if}\ \mathsf{false}\ A\ B \ = \ B$$

$$T \triangleq \forall x{:}\mathsf{bool}.\texttt{if}\ x\ (\$\mathsf{nat} \otimes \mathbf{1})\ (\$\mathsf{bool} \otimes \mathbf{1})$$

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \rightarrow \mathsf{stype} \rightarrow \mathsf{stype} \rightarrow \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B \ = \ A \qquad \texttt{if}\ \mathsf{false}\ A\ B \ = \ B$$

$$T \triangleq \forall x{:}\mathsf{bool}.\texttt{if}\ x\ (\$\mathsf{nat} \otimes \mathbf{1})\ (\$\mathsf{bool} \otimes \mathbf{1})$$

$$\vdash z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{true} \Rightarrow z\langle 23\rangle,\ \mathsf{false} \Rightarrow z\langle \mathsf{true}\rangle) :: z{:}T$$

Richer forms of Dependency

Idea

- All dependencies so far are purely at the level of values
- No way of having protocol **structure** depend on data:
  - "If the received value is OK, receive a String ; otherwise, send a termination message".
- Need type-level functions [TY18].

Boolean-driven Communication

$$\texttt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype}$$

$$\texttt{if}\ \mathsf{true}\ A\ B \ = \ A \qquad \texttt{if}\ \mathsf{false}\ A\ B \ = \ B$$

$$T \triangleq \forall x{:}\mathsf{bool}.\texttt{if}\ x\ (\$\mathsf{nat} \otimes \mathbf{1})\ (\$\mathsf{bool} \otimes \mathbf{1})$$

$$\vdash z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{true} \Rightarrow z\langle 23 \rangle,\ \mathsf{false} \Rightarrow z\langle \mathsf{true} \rangle) :: z{:}T$$

$$\not\vdash z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{false} \Rightarrow z\langle 23 \rangle,\ \mathsf{true} \Rightarrow z\langle \mathsf{true} \rangle) :: z{:}T$$

Outline

Indexing and Decidability

So far:

- Recursive session types
- Indexing + Refinements with decidable (e.g. linear or Presburger) arithmetic.

Indexing and Decidability

So far:

- Recursive session types
- Indexing + Refinements with decidable (e.g. linear or Presburger) arithmetic.
- Type equality?

Indexing and Decidability

So far:

- Recursive session types
- Indexing + Refinements with decidable (e.g. linear or Presburger) arithmetic.
- Type equality? Type bisimulation.
- Decidable in functional (i.e. nominal) settings...

Indexing and Decidability

So far:

- Recursive session types
- Indexing + Refinements with decidable (e.g. linear or Presburger) arithmetic.
- Type equality? Type bisimulation.
- Decidable in functional (i.e. nominal) settings...
- Undecidable in a structural setting [DP20]:
  - Just one type constructor ($\oplus$ or $\&$) is enough.
  - Undecidable with iso or equirecursive types.
  - Undecidable with linear arithmetic + universal prefix quantification.

Indexing and Decidability

So far:

- Recursive session types
- Indexing + Refinements with decidable (e.g. linear or Presburger) arithmetic.
- Type equality? Type bisimulation.
- Decidable in functional (i.e. nominal) settings...
- Undecidable in a structural setting [DP20]:
  - Just one type constructor ($\oplus$ or $\&$) is enough.
  - Undecidable with iso or equirecursive types.
  - Undecidable with linear arithmetic + universal prefix quantification.
- Practical and effective algorithms can be found [DP20]...
- More work to do on this front – nested types [DDMP21], richer dependency [TY18], etc.

Fully-Dependent Type Theory

- What about general dependency instead of just data dependency?

Fully-Dependent Type Theory

- What about general dependency instead of just data dependency?
- Linearity + dependency is a longstanding complex problem.
- Dependency on (quoted) processes studied [TY18], but no inductive/coinductive types.

Fully-Dependent Type Theory

- What about general dependency instead of just data dependency?
- Linearity + dependency is a longstanding complex problem.
- Dependency on (quoted) processes studied [TY18], but no inductive/coinductive types.

Fully-Dependent Type Theory

- What about general dependency instead of just data dependency?
- Linearity + dependency is a longstanding complex problem.
- Dependency on (quoted) processes studied [TY18], but no inductive/coinductive types.
- Decidability of type equality is very subtle.
- Many reasonable notions of process equality (observational, reduction-based, etc.).

Implementation

- All this theory is well and good, but...
- what about **implementations** of refined/dependent session types?
  - Rast [DDP19,DP20] – Resource-aware session types with arithmetic refinements.
  - LiquidPi [GG13] – refinements only on basic data, inference is decidable.
  - Label-dependent session types [TV20] – indexed by naturals, fixed-iteration schema.
  - Session* [ZFHNY20] – multiparty protocol description toolchain, targeting F*
  - STP [NHYA18] – multiparty data refinements in F# type providers.

Wrapping Up

- An (incomplete) overview of 10 years of dependent and logical session types.

Wrapping Up

- An (incomplete) overview of 10 years of dependent and logical session types.
- Logical approach provides a general framework of dependency and indexing.
- Extensible and flexible (proof-carrying code, resource-awareness, etc.)

Wrapping Up

- An (incomplete) overview of 10 years of dependent and logical session types.
- Logical approach provides a general framework of dependency and indexing.
- Extensible and flexible (proof-carrying code, resource-awareness, etc.)
- A bunch of other people got interested in these ideas over the years!

Wrapping Up

- An (incomplete) overview of 10 years of dependent and logical session types.
- Logical approach provides a general framework of dependency and indexing.
- Extensible and flexible (proof-carrying code, resource-awareness, etc.)
- A bunch of other people got interested in these ideas over the years!

Thank you for your time! Questions?