Semantic Subtyping: Challenges, Perspectives, and Open Problems

Giuseppe Castagna

CNRS
École Normale Supérieure de Paris

Outline

1. Motivations and goals.
2. Semantic subtyping.
3. λ-calculus.
4. π-calculus.
5. Some Perspectives.
6. Conclusion

Outline

1. Motivations and goals.
2. Semantic subtyping.
3. λ-calculus.
4. π-calculus.
5. Some Perspectives.
6. Conclusion

Outline

1. Motivations and goals.
2. Semantic subtyping.
3. λ-calculus.
4. π-calculus.
5. Some Perspectives.
6. Conclusion

Outline

1. Motivations and goals.
2. Semantic subtyping.
3. λ-calculus.
4. π-calculus.
5. Some Perspectives.
6. Conclusion

Outline

1. Motivations and goals.
2. Semantic subtyping.
3. λ-calculus.
4. π-calculus.
5. Some Perspectives.
6. Conclusion

Outline

Goal

The goal is to show how to take your favourite type constructors

$$\times,\ \rightarrow,\ \{\ldots\},\ \mathtt{chan}(),\ \ldots$$

and add boolean combinators:

$$\vee,\quad \wedge,\quad \neg$$

so that they behave set-theoretically w.r.t. $\leq$

WHY?

Short answer: they are convenient and you need them to program XML in a typed language with **pattern matching**.

Goal

The goal is to show how to take your favourite type constructors

$$\times,\ \rightarrow,\ \{\dots\},\ \mathtt{chan}(),\ \dots$$

and add boolean combinators:

$$\vee,\quad \wedge,\quad \neg$$

so that they behave set-theoretically w.r.t. $\leq$

WHY?

Short answer: they are convenient and you need them to program XML in a typed language with **pattern matching**.

Goal

The goal is to show how to take your favourite type constructors

$$\times,\ \rightarrow,\ \{\dots\},\ \mathtt{chan}(),\ \dots$$

and add boolean combinators:

$$\vee,\quad \wedge,\quad \neg$$

so that they behave set-theoretically w.r.t. $\leq$

WHY?

Short answer: they are convenient and you need them to program XML in a typed language with **pattern matching**.

Goal

The goal is to show how to take your favourite type constructors

$$\times,\ \rightarrow,\ \{\ldots\},\ \texttt{chan}(),\ \ldots$$

and add boolean combinators:

$$\vee,\quad \wedge,\quad \neg$$

so that they behave set-theoretically w.r.t. $\leq$

WHY?

Short answer: they are convenient and you need them to program XML in a typed language with **pattern matching**.

Why it is difficult?

$$t \ ::= \ B \ \mid \ t{\times}t \ \mid \ t{\rightarrow}t \ \mid \ t{\vee}t \ \mid \ t{\wedge}t \ \mid \ \neg t \ \mid \ \mathbb{0} \ \mid \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \rightarrow t_1 \leq s_2 \rightarrow t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1 \times t_1 \leq s_2 \times t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1 \vee s_2) \rightarrow t \quad \gtrless \quad (s_1 \rightarrow t) \wedge (s_2 \rightarrow t)$$

- Without a clear semantics, subtyping is hard to define.

Why it is difficult?

$$t \ ::= \ B \ \mid \ t \times t \ \mid \ t \to t \ \mid \ t \vee t \ \mid \ t \wedge t \ \mid \ \neg t \ \mid \ \mathbb{0} \ \mid \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \to t_1 \leq s_2 \to t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1 \times t_1 \leq s_2 \times t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1 \vee s_2) \to t \ \gtrless \ (s_1 \to t) \wedge (s_2 \to t)$$

- Without a clear semantics, subtyping is hard to define.

Why it is difficult?

$$t \ ::= \ B \ \mid \ t \times t \ \mid \ t \rightarrow t \ \mid \ t \vee t \ \mid \ t \wedge t \ \mid \ \neg t \ \mid \ \mathbb{0} \ \mid \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \rightarrow t_1 \leq s_2 \rightarrow t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1 \times t_1 \leq s_2 \times t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1 \vee s_2) \rightarrow t \quad \gtrless \quad (s_1 \rightarrow t) \wedge (s_2 \rightarrow t)$$

- Without a clear semantics, subtyping is hard to define.

Why it is difficult?

$$t \ ::= \ B \ | \ t \times t \ | \ t \to t \ | \ t \vee t \ | \ t \wedge t \ | \ \neg t \ | \ \mathbb{0} \ | \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \to t_1 \leq s_2 \to t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1 \times t_1 \leq s_2 \times t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1 \vee s_2) \to t \quad \gtrless \quad (s_1 \to t) \wedge (s_2 \to t)$$

- Without a clear semantics, subtyping is hard to define.

Why it is difficult?

$$t \ ::= \ B \ \mid \ t \times t \ \mid \ t \to t \ \mid \ t \vee t \ \mid \ t \wedge t \ \mid \ \neg t \ \mid \ \mathbb{0} \ \mid \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \to t_1 \leq s_2 \to t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1 \times t_1 \leq s_2 \times t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1 \vee s_2) \to t \quad \gtreqless \quad (s_1 \to t) \wedge (s_2 \to t)$$

- Without a clear semantics, subtyping is hard to define.

Why it is difficult?

$$t \;::=\; B \;\mid\; t{\times}t \;\mid\; t{\rightarrow}t \;\mid\; t{\vee}t \;\mid\; t{\wedge}t \;\mid\; \neg t \;\mid\; \mathbb{0} \;\mid\; \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \rightarrow t_1 \leq s_2 \rightarrow t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1{\times}t_1 \leq s_2{\times}t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1{\vee}s_2) \rightarrow t \quad \gtrless \quad (s_1 \rightarrow t){\wedge}(s_2 \rightarrow t)$$

- Without a clear semantics, subtyping is hard to define, e.g.

$$ch^{+}(s){\wedge}ch^{-}(t) \quad \leq \quad ch^{-}(s){\vee}ch^{+}(t) \quad ???$$

Why it is difficult?

$$t \ ::= \ B \ | \ t{\times}t \ | \ t{\to}t \ | \ t{\vee}t \ | \ t{\wedge}t \ | \ \neg t \ | \ \mathbb{0} \ | \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \to t_1 \leq s_2 \to t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1{\times}t_1 \leq s_2{\times}t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1{\vee}s_2) \to t \quad \gtrless \quad (s_1 \to t){\wedge}(s_2 \to t)$$

- Without a clear semantics, subtyping is hard to define, e.g.

$$ch^{+}(s){\wedge}ch^{-}(t) \quad \leq \quad ch^{-}(s){\vee}ch^{+}(t) \quad ???$$

Why it is difficult?

$$t \ ::= \ B \ | \ t{\times}t \ | \ t{\rightarrow}t \ | \ t{\vee}t \ | \ t{\wedge}t \ | \ \neg t \ | \ \mathbb{0} \ | \ \mathbb{1}$$

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \rightarrow t_1 \leq s_2 \rightarrow t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1{\times}t_1 \leq s_2{\times}t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1{\vee}s_2) \rightarrow t \quad \substack{\geq \\ \leq} \quad (s_1 \rightarrow t){\wedge}(s_2 \rightarrow t)$$

- Without a clear semantics, subtyping is hard to define, e.g.

$$ch^{+}(s){\wedge}ch^{-}(t) \quad \leq \quad ch^{-}(s){\vee}ch^{+}(t) \quad ???$$

Why it is difficult?

MAIN IDEA

Instead of defining the subtyping relation so that it conforms to the semantic of types, define the semantics of types and derive the subtyping relation.

- Handling subtyping without combinators is easy: constructors do not mix, e.g.:

$$\frac{s_2 \leq s_1 \qquad t_1 \leq t_2}{s_1 \to t_1 \leq s_2 \to t_2} \qquad\qquad \frac{s_1 \leq s_2 \qquad t_1 \leq t_2}{s_1 \times t_1 \leq s_2 \times t_2}$$

- With combinators is much harder: combinators distribute over constructors, e.g.

$$(s_1 \vee s_2) \to t \quad \substack{\geq \\ \leq} \quad (s_1 \to t) \wedge (s_2 \to t)$$

- Without a clear semantics, subtyping is hard to define, e.g.

$$ch^{+}(s) \wedge ch^{-}(t) \quad \leq \quad ch^{-}(s) \vee ch^{+}(t) \quad ???$$

Why it is difficult?

MAIN IDEA

Instead of defining the subtyping relation so that it conforms to the semantic of types, define the semantics of types and derive the subtyping relation.

- **Not a particularly new idea**. Many attempts (e.g. Aiken&Wimmers, Damm,. . . , Hosoya&Pierce).
- **None fully satisfactory.** (no negation, or no function types, or restrictions on unions and intersections, . . .)
- **Starting point of what follows: the approach of Hosoya&Pierce.**

Why it is difficult?

MAIN IDEA

Instead of defining the subtyping relation so that it conforms to the semantic of types, define the semantics of types and derive the subtyping relation.

- **Not a particularly new idea**. Many attempts (e.g. Aiken&Wimmers, Damm,. . . , Hosoya&Pierce).
- **None fully satisfactory.** (no negation, or no function types, or restrictions on unions and intersections, . . .)
- **Starting point of what follows: the approach of Hosoya&Pierce.**

Why it is difficult?

MAIN IDEA

Instead of defining the subtyping relation so that it conforms to the semantic of types, define the semantics of types and derive the subtyping relation.

- **Not a particularly new idea**. Many attempts (e.g. Aiken&Wimmers, Damm,. . . , Hosoya&Pierce).
- **None fully satisfactory.** (no negation, or no function types, or restrictions on unions and intersections, . . .)
- **Starting point of what follows: the approach of Hosoya&Pierce.**

Semantic subtyping

Semantic subtyping

1. Define a set-theoretic semantics of the types:

$$[\![\ ]\!] : \mathbf{Types} \longrightarrow \mathscr{P}(\mathscr{D})$$

2. Define the subtyping relation as follows:

$$s \leq t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

KEY OBSERVATION 1:

The *model of types* may be independent from a *model of terms*

Hosoya and Pierce use the model of values:

$$[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

Works because XML documents are the only XDuce values and for them $[\![t]\!]_{\mathscr{V}}$ can be defined independently from the typing relation

Semantic subtyping

1. Define a set-theoretic semantics of the types:

$$[\![\ ]\!] : \textbf{Types} \longrightarrow \mathscr{P}(\mathscr{D})$$

2. Define the subtyping relation as follows:

$$s \leq t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

KEY OBSERVATION 1:

The *model of types* may be independent from a *model of terms*

Hosoya and Pierce use the model of values:

$$[\![t]\!]_{\mathscr{V}} = \{ v \mid \vdash v : t \}$$

Works because XML documents are the only XDuce values and for them $[\![t]\!]_{\mathscr{V}}$ can be defined independently from the typing relation

Semantic subtyping

1. Define a set-theoretic semantics of the types:

$$[\![\ ]\!] : \mathbf{Types} \longrightarrow \mathscr{P}(\mathscr{D})$$

2. Define the subtyping relation as follows:

$$s \leq t \quad \overset{def}{\iff} \quad [\![s]\!] \subseteq [\![t]\!]$$

KEY OBSERVATION 1:

The *model of types* may be independent from a *model of terms*

Hosoya and Pierce use the model of values:

$$[\![t]\!]_{\mathscr{V}} = \{v \mid \ \vdash v : t\}$$

Works because XML documents are the only XDuce values and for them $[\![t]\!]_{\mathscr{V}}$ can be defined independently from the typing relation

Semantic subtyping

1. Define a set-theoretic semantics of the types:

$$[\![\ ]\!] : \textbf{Types} \longrightarrow \mathscr{P}(\mathscr{D})$$

2. Define the subtyping relation as follows:

$$s \leq t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

KEY OBSERVATION 1:

The *model of types* may be independent from a *model of terms*

Hosoya and Pierce use the model of values:

$$[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

Works because XML documents are the only XDuce values and for them $[\![t]\!]_{\mathscr{V}}$ can be defined independently from the typing relation

Semantic subtyping

1. Define a set-theoretic semantics of the types:

$$[\![\ ]\!] : \mathbf{Types} \longrightarrow \mathscr{P}(\mathscr{D})$$

2. Define the subtyping relation as follows:

$$s \leq t \quad \stackrel{def}{\iff} \quad [\![s]\!] \subseteq [\![t]\!]$$

KEY OBSERVATION 1:

The *model of types* may be independent from a *model of terms*

Hosoya and Pierce use the model of values:

$$[\![t]\!]_{\mathscr{V}} = \{v \mid \ \vdash v : t\}$$

Works because XML documents are the only XDuce values and for them $[\![t]\!]_{\mathscr{V}}$ can be defined independently from the typing relation

Semantic subtyping

1. Define a set-theoretic semantics of the types:

$$[\![\ ]\!] : \textbf{Types} \longrightarrow \mathscr{P}(\mathscr{D})$$

2. Define the subtyping relation as follows:

$$s \leq t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

KEY OBSERVATION 1:

The *model of types* may be independent from a *model of terms*

Hosoya and Pierce use the model of values:

$$[\![t]\!]_{\mathscr{V}} = \{v \mid \ \vdash v : t\}$$

Works because XML documents are the only XDuce values and for them $[\![t]\!]_{\mathscr{V}}$ can be defined independently from the typing relation

Circularity

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

Circularity

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

Circularity

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$[\![t]\!]_{\mathscr{V}}$

Circularity

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd} {[\![t]\!]_{\mathscr{V}}} \arrow[d, bend left] \\ \vdash v : t \end{tikzcd}$$

Circularity

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid\ \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd} & & {[\![t]\!]_{\mathscr{V}}} \arrow[d, bend left] \\ \vdash e : t & & \vdash v : t \arrow[ll, bend left] \end{tikzcd}$$

Circularity

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid \ \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd}
t \leq t & & [\![t]\!]_{\mathscr{V}} \arrow[d, bend left] \\
\vdash e : t \arrow[u, bend left] & & \vdash v : t \arrow[ll, bend left]
\end{tikzcd}$$

Circularity

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd} t \leq t \arrow[r, bend left] & {[\![t]\!]_{\mathscr{V}}} \arrow[d, bend left] \\ \vdash e : t \arrow[u, bend left] & \vdash v : t \arrow[l, bend left] \end{tikzcd}$$

Circularity

Model of values

$$t \leq s \iff \llbracket t \rrbracket_{\mathscr{V}} \subseteq \llbracket s \rrbracket_{\mathscr{V}} \quad \text{where} \quad \llbracket t \rrbracket_{\mathscr{V}} = \{v \mid\ \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd}
t \leq t \arrow[r, bend left] & \llbracket t \rrbracket_{\mathscr{V}} \arrow[d, bend left] \\
\vdash e : t & \vdash v : t \arrow[ul, bend left]
\end{tikzcd}$$

A similar circularity holds for π-calculus channels, as well

Bootstrap

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

Bootstrap

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$[\![t]\!]_{\mathscr{D}}$

$t \leq t \qquad [\![t]\!]_{\mathscr{V}}$

$\vdash e : t \qquad \vdash v : t$

Bootstrap

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$[\![t]\!]_{\mathscr{D}}$$

$$t \leq t \qquad [\![t]\!]_{\mathscr{V}}$$

$$\vdash e : t \qquad \vdash v : t$$

Bootstrap

Model of values

$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}}$ where $[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd}
 & {[\![t]\!]_{\mathscr{D}}} \\
t \leq t \arrow[ur, bend left] & {[\![t]\!]_{\mathscr{V}}} \\
\vdash e : t & \vdash v : t
\end{tikzcd}$$

Bootstrap

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid\ \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd}
 & {[\![t]\!]_{\mathscr{D}}} \\
t \leq t \arrow[ur, bend left] & {[\![t]\!]_{\mathscr{V}}} \\
\vdash e : t \arrow[u, bend left] & \vdash v : t
\end{tikzcd}$$

Bootstrap

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd}
 & {[\![t]\!]_{\mathscr{D}}} \\
t \leq t \arrow[ur, bend left] & {[\![t]\!]_{\mathscr{V}}} \\
\vdash e : t \arrow[u, bend left] & \vdash v : t \arrow[l, bend left]
\end{tikzcd}$$

Bootstrap

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

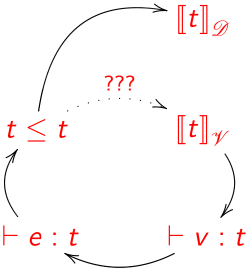
$$\begin{tikzcd}
 & {[\![t]\!]_{\mathscr{D}}} \\
t \leq t \arrow[ur, bend left] & {[\![t]\!]_{\mathscr{V}}} \arrow[d, bend left] \\
\vdash e : t \arrow[u, bend left] & \vdash v : t \arrow[l, bend left]
\end{tikzcd}$$

Bootstrap

Model of values

$$t \leq s \iff [\![t]\!]_{\mathscr{V}} \subseteq [\![s]\!]_{\mathscr{V}} \quad \text{where} \quad [\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined



Bootstrap

Model of values

$t \leq s \iff \llbracket t \rrbracket_{\mathscr{V}} \subseteq \llbracket s \rrbracket_{\mathscr{V}}$ where $\llbracket t \rrbracket_{\mathscr{V}} = \{v \mid\ \vdash v : t\}$

For instance, it does not work with arrow types: values are λ-abstractions and need (sub)typing to be defined

$$\begin{tikzcd}
 & \llbracket t \rrbracket_{\mathscr{D}} \arrow[d, equal] \\
t \leq t \arrow[ur, bend left] \arrow[r, dotted, "???"] & \llbracket t \rrbracket_{\mathscr{V}} \arrow[d, bend left] \\
\vdash e : t \arrow[u, bend left] & \vdash v : t \arrow[l, bend left]
\end{tikzcd}$$

Semantic subtyping in five steps:

1. **Add boolean combinators:** $\vee$, $\wedge$, $\neg$
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, …)
2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}}$: **Types** $\rightarrow \mathscr{P}(\mathscr{D})$
   $[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$, $[\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$, $[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}}$
   $$s \leq_{\mathscr{D}} t \stackrel{def}{\iff} [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$
3. Find a subtyping algorithm by using the set-theoretic properties of the model
4. Define a language and type it by using $s \leq_{\mathscr{D}} t$:
   $$\frac{\Gamma \vdash_{\mathscr{D}} e : s \quad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$
5. Close the circle: define the types-as-set-of-values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$ and check
   $$s \leq_{\mathscr{D}} t \iff s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction, …

Semantic subtyping in five steps:

1. **Add boolean combinators: $\vee$, $\wedge$, $\neg$**
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, ...)

2. **Define a set-theoretic semantics: $[\![\,]\!]_{\mathscr{D}}$: Types $\rightarrow \mathscr{P}(\mathscr{D})$**
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}\ ,\ [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}\ ,\ [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$

The rest of the story is standard: subject reduction...

Semantic subtyping in five steps:

1. **Add boolean combinators: $\vee$, $\wedge$, $\neg$**
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, $\ldots$)

2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}}$: **Types** $\rightarrow \mathscr{P}(\mathscr{D})$
   $([\![s\wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \;,\; [\![s\vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \;,\; [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$

   $$s \leq_{\mathscr{D}} t \;\stackrel{def}{\Longleftrightarrow}\; [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. Find a subtyping algorithm by using the set-theoretic properties of the model.

4. Define a language and type it by using $s \leq_{\mathscr{D}} t$:

   $$\frac{\Gamma \vdash_{\mathscr{D}} e : s \quad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. Close the circle: define the types-as-set-of-values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$ and check

   $$s \leq_{\mathscr{D}} t \iff s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction, . . .

Semantic subtyping in five steps:

1. **Add boolean combinators: $\vee$, $\wedge$, $\neg$**
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, ...)

2. **Define a set-theoretic semantics: $[\![\;]\!]_{\mathscr{D}}$: Types $\rightarrow \mathscr{P}(\mathscr{D})$**
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \;\;,\;\; [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \;\;,\;\; [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$

$$s \leq_{\mathscr{D}} t \;\stackrel{def}{\iff}\; [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. Find a subtyping algorithm by using the set-theoretic properties of the model [optional but advisable]

4. Define a language and type it by using $s \leq_{\mathscr{D}} t$:

$$\frac{\Gamma \vdash_{\mathscr{D}} e : s \quad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. Close the circle: define the types as set of values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$ and check

$$s \leq_{\mathscr{D}} t \iff s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction, . . .

Semantic subtyping in five steps:

1. **Add boolean combinators: $\vee$, $\wedge$, $\neg$**
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, $\ldots$)

2. **Define a set-theoretic semantics: $[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$**
   ($[\![s\wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$, $[\![s\vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$, $[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}}$)

$$s \leq_{\mathscr{D}} t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]

4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:

The rest of the story is standard: subject reduction...

Semantic subtyping in five steps:

1. **Add boolean combinators:** $\vee$, $\wedge$, $\neg$
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, ...)
2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \;\;,\;\; [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \;\;,\;\; [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$
   $$s \leq_{\mathscr{D}} t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$
3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]
4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:
   $$\frac{\Gamma \vdash_{\mathscr{D}} e : s \quad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

The rest of the story is standard: subject reduction...

Semantic subtyping in five steps:

1. **Add boolean combinators:** $\vee$, $\wedge$, $\neg$
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, …)

2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \;,\; [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \;,\; [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$
   $$s \leq_{\mathscr{D}} t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]

4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:
   $$\frac{\Gamma \vdash_{\mathscr{D}} e : s \qquad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. **Close the circle:** define the types-as-set-of-values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$ and check
   $$s \leq_{\mathscr{D}} t \quad \Longleftrightarrow \quad s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction, …

Semantic subtyping in five steps:

1. **Add boolean combinators:** $\vee$, $\wedge$, $\neg$
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, $\ldots$)

2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}\ ,\ [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}\ ,\ [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$

$$s \leq_{\mathscr{D}} t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]

4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:

$$\frac{\Gamma \vdash_{\mathscr{D}} e : s \qquad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. **Close the circle:** define the types-as-set-of-values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$ and check

$$s \leq_{\mathscr{D}} t \iff s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction, ...

Semantic subtyping in five steps:

1. **Add boolean combinators:** $\vee$, $\wedge$, $\neg$
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, ...)

2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$
   $([\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}\ ,\ [\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}\ ,\ [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$

$$s \leq_{\mathscr{D}} t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]

4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:

$$\frac{\Gamma \vdash_{\mathscr{D}} e : s \qquad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. **Close the circle:** define the types-as-set-of-values semantics
   $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid\ \vdash_{\mathscr{D}} v : t\}$ and check

$$s \leq_{\mathscr{D}} t \quad \Longleftrightarrow \quad s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction ...

Semantic subtyping in five steps:

1. **Add boolean combinators: $\vee$, $\wedge$, $\neg$**
   to your favourite type *constructors* (e.g., $\rightarrow$, $\times$, $ch(\,)$, ...)

2. **Define a set-theoretic semantics:** $[\![\;]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}\;,\;[\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}\;,\;[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$

$$s \leq_{\mathscr{D}} t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]

4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:

$$\frac{\Gamma \vdash_{\mathscr{D}} e : s \qquad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. **Close the circle:** define the types-as-set-of-values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \;\vdash_{\mathscr{D}} v : t\}$ and check

$$s \leq_{\mathscr{D}} t \quad \Longleftrightarrow \quad s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction...

Semantic subtyping in five steps:

1. **Add boolean combinators:** $\vee$, $\wedge$, $\neg$
   to your favourite type *constructors* (e.g., $\to$, $\times$, $ch(\,)$, ...)

2. **Define a set-theoretic semantics:** $[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$
   $([\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}\ ,\ [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}\ ,\ [\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}})$
   $$s \leq_{\mathscr{D}} t \quad \stackrel{def}{\Longleftrightarrow} \quad [\![s]\!]_{\mathscr{D}} \subseteq [\![t]\!]_{\mathscr{D}}$$

3. **Find a subtyping algorithm** by using the set-theoretic properties of the model [optional but advisable]

4. **Define a language and type it** by using $s \leq_{\mathscr{D}} t$:
   $$\frac{\Gamma \vdash_{\mathscr{D}} e : s \qquad s \leq_{\mathscr{D}} t}{\Gamma \vdash_{\mathscr{D}} e : t}$$

5. **Close the circle:** define the types-as-set-of-values semantics $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid\ \vdash_{\mathscr{D}} v : t\}$ and check
   $$s \leq_{\mathscr{D}} t \qquad \Longleftrightarrow \qquad s \leq_{\mathscr{V}} t$$

The rest of the story is standard: subject reduction, ...

λ-calculus.

STEP 1: types for λ

Types	t	::=	b	basic types	type constructors
		|	$t \times t$	product type	
		|	$t \rightarrow t$	function type	
		|	$\mathbb{0}$	empty type	type combinators
		|	$\mathbb{1}$	top type	
		|	$\neg t$	negation type	
		|	$t \vee t$	union type	
		|	$t \wedge t$	intersection type	

STEP 1: types for λ

Types	t	::=	b	basic types	type constructors
		|	$t \times t$	product type	
		|	$t \to t$	function type	
		|	$\mathbb{0}$	empty type	type combinators
		|	$\mathbb{1}$	top type	
		|	$\neg t$	negation type	
		|	$t \vee t$	union type	
		|	$t \wedge t$	intersection type	

STEP 1: types for λ

$$
\textbf{Types} \quad t ::= \left.\begin{array}{lll} & b & \text{basic types} \\ \mid & t\times t & \text{product type} \\ \mid & t\rightarrow t & \text{function type} \end{array}\right\} \text{type constructors}
$$

$$
\left.\begin{array}{lll} \mid & \mathbb{0} & \text{empty type} \\ \mid & \mathbb{1} & \text{top type} \\ \mid & \neg t & \text{negation type} \\ \mid & t \vee t & \text{union type} \\ \mid & t \wedge t & \text{intersection type} \end{array}\right\} \text{type combinators}
$$

STEP 2: set-theoretic model

$$[\![\ ]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:

$[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$ $\quad$ $[\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}}$ $\quad$ $[\![s \times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\ ]\!]$ that behaves w.r.t. $\subseteq$ as if equation $(*)$ held, namely

$$[\![t_1 \to s_1]\!] \subseteq [\![t_2 \to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\;]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:

$$[\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \qquad [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$$

$$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}} \qquad [\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\;]\!]$ that behaves w.r.t. $\subseteq$ **as if** equation $(*)$ held, namely

$$[\![t_1{\to}s_1]\!] \subseteq [\![t_2{\to}s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\ ]\!]_{\mathscr{D}} : \textbf{Types} \rightarrow \mathscr{P}(\mathscr{D})$$

Easy part:
$[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$ $\quad [\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$
$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}}$ $\quad [\![s \times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$

Hard part: $[\![t \rightarrow s]\!] = ???$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\ ]\!]$ that behaves w.r.t. $\subseteq$ **as if** equation $(*)$ held, namely

$$[\![t_1 \rightarrow s_1]\!] \subseteq [\![t_2 \rightarrow s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\,]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:
$$[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \qquad [\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$$
$$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}} \qquad [\![s \times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$$

Hard part: $[\![t \to s]\!] = \{\text{functions from } [\![t]\!] \text{ to } [\![s]\!]\}$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\,]\!]$ that behaves w.r.t. $\subseteq$ as if equation $(*)$ held, namely

$$[\![t_1 \to s_1]\!] \subseteq [\![t_2 \to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\ ]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part: $[\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$ $\quad [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$
$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}}$ $\quad [\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$

Hard part: $[\![t{\to}s]\!] = \{f \subseteq \mathscr{D}^2 \mid \forall (d_1,d_2) \in f.\ d_1 \in [\![t]\!] \Rightarrow d_2 \in [\![s]\!]\}$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\ ]\!]$ that behaves w.r.t. $\subseteq$ **as if** equation $(*)$ held, namely

$$[\![t_1{\to}s_1]\!] \subseteq [\![t_2{\to}s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\ ]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:

$$[\![s\wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \qquad [\![s\vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$$

$$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}} \qquad [\![s\times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$$

Hard part: $[\![t\to s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $\qquad (*)$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\ ]\!]$ that behaves w.r.t. $\subseteq$ as if equation $(*)$ held, namely

$$[\![t_1\to s_1]\!] \subseteq [\![t_2\to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\;]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:

$$[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \qquad [\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$$

$$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}} \qquad [\![s \times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$$

Hard part: $[\![t \to s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $\qquad (*)$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

Use any $[\![\;]\!]$ that behaves w.r.t. $\subseteq$ as if equation $(*)$ held, namely

$$[\![t_1 \to s_1]\!] \subseteq [\![t_2 \to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\;]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part: $[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$ $\quad [\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$
$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}}$ $\quad [\![s \times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$

Hard part: $[\![t \to s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $\qquad (*)$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

We need the model to state **how types are related** rather than **what the types are**

Use any $[\![\;]\!]$ that behaves w.r.t. $\subseteq$ **as if** equation $(*)$ held, namely

$$[\![t_1 \to s_1]\!] \subseteq [\![t_2 \to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\ ]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:

$$[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \qquad [\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$$

$$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}} \qquad [\![s \times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$$

Hard part: $[\![t \to s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $\qquad (*)$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

We need the model to state **how types are related** rather than **what the types are**

Use any $[\![\ ]\!]$ that behaves w.r.t. $\subseteq$ **as if** equation $(*)$ held, namely

$$[\![t_1 \to s_1]\!] \subseteq [\![t_2 \to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

$$[\![\ ]\!]_{\mathscr{D}} : \textbf{Types} \to \mathscr{P}(\mathscr{D})$$

Easy part:

$$[\![s\wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}} \qquad [\![s\vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$$

$$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \setminus [\![t]\!]_{\mathscr{D}} \qquad [\![s\times t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \times [\![t]\!]_{\mathscr{D}}$$

Hard part: $[\![t\to s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $\quad (*)$

Impossible since it requires $\mathscr{P}(\mathscr{D}^2) \subseteq \mathscr{D}$

KEY OBSERVATION 2:

We need the model to state **how types are related** rather than **what the types are**

Use any $[\![\ ]\!]$ that behaves w.r.t. $\subseteq$ **as if** equation $(*)$ held, namely

$$[\![t_1\to s_1]\!] \subseteq [\![t_2\to s_2]\!] \iff \mathscr{P}(\overline{[\![t_1]\!] \times \overline{[\![s_1]\!]}}) \subseteq \mathscr{P}(\overline{[\![t_2]\!] \times \overline{[\![s_2]\!]}})$$

STEP 2: set-theoretic model

Solution: $[\![t{\to}s]\!] = \mathscr{P}_f(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $(*)$

Subtyping is completely characterised by type **emptiness**

Indeed: $s \leq t \Leftrightarrow [\![s]\!] \subseteq [\![t]\!] \Leftrightarrow [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \Leftrightarrow [\![s{\wedge}\neg t]\!] = \varnothing$

$$\mathscr{P}_f(X) = \varnothing \iff \mathscr{P}(X) = \varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping as

$$[\![t{\to}s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}} = \emptyset$ $[\![\mathbb{1}]\!]_{\mathscr{D}} = \mathscr{D}$ $[\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$ $[\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \backslash [\![t]\!]_{\mathscr{D}}$ $[\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!] \times [\![t]\!]$ $[\![t{\to}s]\!]_{\mathscr{D}} = \mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}} \times \overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 2: set-theoretic model

Solution: $[\![t{\to}s]\!] = \mathscr{P}_f(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $(*)$

Subtyping is completely characterised by type **emptiness**

Indeed: $s \leq t \Leftrightarrow [\![s]\!] \subseteq [\![t]\!] \Leftrightarrow [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \Leftrightarrow [\![s{\wedge}\neg t]\!] = \varnothing$

$$\mathscr{P}_f(X) = \varnothing \iff \mathscr{P}(X) = \varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping *as*

$$[\![t{\to}s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}} = \emptyset \quad [\![\mathbb{1}]\!]_{\mathscr{D}} = \mathscr{D} \quad [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \quad [\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \backslash [\![t]\!]_{\mathscr{D}} \quad [\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!] \times [\![t]\!] \quad [\![t{\to}s]\!]_{\mathscr{D}} = \mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}} \times \overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 2: set-theoretic model

Solution: $[\![t{\longrightarrow}s]\!] = \mathscr{P}_f(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $\quad(*)$

Subtyping is completely characterised by type **emptiness**

Indeed: $s \leq t \Leftrightarrow [\![s]\!] \subseteq [\![t]\!] \Leftrightarrow [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \Leftrightarrow [\![s{\wedge}\neg t]\!] = \varnothing$

$$\mathscr{P}_f(X) = \varnothing \iff \mathscr{P}(X) = \varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping as

$$[\![t{\longrightarrow}s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}} = \emptyset \quad [\![\mathbb{1}]\!]_{\mathscr{D}} = \mathscr{D} \quad [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \quad [\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \backslash [\![t]\!]_{\mathscr{D}} \quad [\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!] \times [\![t]\!] \quad [\![t{\longrightarrow}s]\!]_{\mathscr{D}} = \mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}} \times \overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 2: set-theoretic model

Solution: $$[\![t{\to}s]\!] = \mathscr{P}_f(\overline{[\![t]\!]\times\overline{[\![s]\!]}}) \qquad (*)$$

Subtyping is completely characterised by type **emptiness**

Indeed: $s \leq t \Leftrightarrow [\![s]\!] \subseteq [\![t]\!] \Leftrightarrow [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \Leftrightarrow [\![s{\wedge}\neg t]\!] = \varnothing$

$$\mathscr{P}_f(X) = \varnothing \iff \mathscr{P}(X) = \varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping *as*

$$[\![t{\to}s]\!] = \mathscr{P}(\overline{[\![t]\!]\times\overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}} = \emptyset \quad [\![\mathbb{1}]\!]_{\mathscr{D}} = \mathscr{D} \quad [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \quad [\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \backslash [\![t]\!]_{\mathscr{D}} \quad [\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!] \times [\![t]\!] \quad [\![t{\to}s]\!]_{\mathscr{D}} = \mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}} \times \overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 2: set-theoretic model

Solution: $$[\![t{\to}s]\!] = \mathscr{P}_f(\overline{[\![t]\!] \times \overline{[\![s]\!]}}) \qquad (*)$$

Subtyping is completely characterised by type **emptiness**

Indeed: $s \leq t \Leftrightarrow [\![s]\!] \subseteq [\![t]\!] \Leftrightarrow [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \Leftrightarrow [\![s{\wedge}\neg t]\!] = \varnothing$

$$\mathscr{P}_f(X) = \varnothing \iff \mathscr{P}(X) = \varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping *as*

$$[\![t{\to}s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}} = \emptyset \quad [\![\mathbb{1}]\!]_{\mathscr{D}} = \mathscr{D} \quad [\![s{\vee}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}} \quad [\![s{\wedge}t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \backslash [\![t]\!]_{\mathscr{D}} \quad [\![s{\times}t]\!]_{\mathscr{D}} = [\![s]\!] \times [\![t]\!] \quad [\![t{\to}s]\!]_{\mathscr{D}} = \mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}} \times \overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 2: set-theoretic model

Solution: $$[\![t\to s]\!] = \mathscr{P}_f(\overline{[\![t]\!]\times\overline{[\![s]\!]}}) \qquad (*)$$

Subtyping is completely characterised by type **emptiness**

Indeed: $s\leq t \Leftrightarrow [\![s]\!]\subseteq[\![t]\!] \Leftrightarrow [\![s]\!]\cap\overline{[\![t]\!]}=\varnothing \Leftrightarrow [\![s\wedge\neg t]\!]=\varnothing$

$$\mathscr{P}_f(X)=\varnothing \iff \mathscr{P}(X)=\varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping *as*

$$[\![t\to s]\!] = \mathscr{P}(\overline{[\![t]\!]\times\overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}}=\emptyset \quad [\![\mathbb{1}]\!]_{\mathscr{D}}=\mathscr{D} \quad [\![s\vee t]\!]_{\mathscr{D}}=[\![s]\!]_{\mathscr{D}}\cup[\![t]\!]_{\mathscr{D}} \quad [\![s\wedge t]\!]_{\mathscr{D}}=[\![s]\!]_{\mathscr{D}}\cap[\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}}=\mathscr{D}\setminus[\![t]\!]_{\mathscr{D}} \quad [\![s\times t]\!]_{\mathscr{D}}=[\![s]\!]\times[\![t]\!] \quad [\![t\to s]\!]_{\mathscr{D}}=\mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}}\times\overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 2: set-theoretic model

Solution: $[\![t \to s]\!] = \mathscr{P}_f(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$ $(*)$

Subtyping is completely characterised by type **emptiness**

Indeed: $s \leq t \Leftrightarrow [\![s]\!] \subseteq [\![t]\!] \Leftrightarrow [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \Leftrightarrow [\![s \wedge \neg t]\!] = \varnothing$

$$\mathscr{P}_f(X) = \varnothing \iff \mathscr{P}(X) = \varnothing$$

Therefore, $(*)$ induces the same subtyping relation subtyping *as*

$$[\![t \to s]\!] = \mathscr{P}(\overline{[\![t]\!] \times \overline{[\![s]\!]}})$$

Bootstrap model

$\mathscr{D}$ least solution of $X = X^2 + \mathscr{P}_f(X^2)$

$[\![\mathbb{0}]\!]_{\mathscr{D}} = \emptyset$ $[\![\mathbb{1}]\!]_{\mathscr{D}} = \mathscr{D}$ $[\![s \vee t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cup [\![t]\!]_{\mathscr{D}}$ $[\![s \wedge t]\!]_{\mathscr{D}} = [\![s]\!]_{\mathscr{D}} \cap [\![t]\!]_{\mathscr{D}}$

$[\![\neg t]\!]_{\mathscr{D}} = \mathscr{D} \backslash [\![t]\!]_{\mathscr{D}}$ $[\![s \times t]\!]_{\mathscr{D}} = [\![s]\!] \times [\![t]\!]$ $[\![t \to s]\!]_{\mathscr{D}} = \mathscr{P}_f(\overline{[\![t]\!]_{\mathscr{D}} \times \overline{[\![s]\!]_{\mathscr{D}}}})$

STEP 3: Subtyping algorithm

- Define:

$$s \leq t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

- Use it to deduce some subtyping relations, e.g.

$$(t_1 \vee t_2) \rightarrow (s_1 \wedge s_2) \;\lneq\; (t_1 \rightarrow s_1) \wedge (t_2 \rightarrow s_2) \;\lneq\; (t_1 \vee t_2) \rightarrow (s_1 \vee s_2)$$

- How to decide $s \leq t$ in general?

STEP 3: Subtyping algorithm

- Define:

$$s \leq t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

- Use it to deduce some subtyping relations, e.g.

$$(t_1 \vee t_2) \rightarrow (s_1 \wedge s_2) \ \lneq \ (t_1 \rightarrow s_1) \wedge (t_2 \rightarrow s_2) \ \lneq \ (t_1 \vee t_2) \rightarrow (s_1 \vee s_2)$$

- How to decide $s \leq t$ in general?

STEP 3: Subtyping algorithm

- Define:

$$s \leq t \quad \overset{def}{\Longleftrightarrow} \quad [\![s]\!] \subseteq [\![t]\!]$$

- Use it to deduce some subtyping relations, e.g.

$$(t_1 \vee t_2) \rightarrow (s_1 \wedge s_2) \;\lneq\; (t_1 \rightarrow s_1) \wedge (t_2 \rightarrow s_2) \;\lneq\; (t_1 \vee t_2) \rightarrow (s_1 \vee s_2)$$

- How to decide $s \leq t$ in general?

STEP 3: Subtyping algorithm

Some ugly formulae:

$$\bigwedge_{i\in I} t_i\times s_i \leq \bigvee_{i\in J} t_i\times s_i$$

$$\iff \forall J'\subseteq J.\ \left(\bigwedge_{i\in I} t_i \leq \bigvee_{i\in J'} t_i\right) \text{ or } \left(\bigwedge_{i\in I} s_i \leq \bigvee_{i\in J\setminus J'} s_i\right)$$

$$\bigwedge_{i\in I} t_i\rightarrow s_i \leq \bigvee_{i\in J} t_i\rightarrow s_i$$

$$\iff \exists j\in J.\forall I'\subseteq I.\ \left(t_j \leq \bigvee_{i\in I'} t_i\right) \text{ or } \left(I'\neq I \text{ et } \bigwedge_{i\in I\setminus I'} s_i \leq s_j\right)$$

STEP 3: Subtyping algorithm

$$s \leq t?$$

Recall that:

$$s \leq t \iff [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \iff [\![s \wedge \neg t]\!] = \varnothing \iff s \wedge \neg t = \mathbb{0}$$

1. Consider $s \wedge \neg t$
2. Put it in canonical form

$$\bigvee_{(P,N)\in\Pi} \Big(\big(\bigwedge_{s\times t \in P} s\times t\big) \wedge \big(\bigwedge_{s\times t\in N} \neg(s\times t)\big)\Big) \quad \bigvee_{(P,N)\in\Sigma} \Big(\big(\bigwedge_{s\to t \in P} s\to t\big) \wedge \big(\bigwedge_{s\to t\in N} \neg(s\to t)\big)\Big)$$

3. Decide (coinductively) whether the two summands are both empty by applying the ugly formulae of the previous slide.

STEP 3: Subtyping algorithm

$$s \leq t?$$

Recall that:

$$s \leq t \iff [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \iff [\![s\wedge\neg t]\!] = \varnothing \iff s\wedge\neg t = \mathbb{0}$$

1. Consider $s\wedge\neg t$
2. Put it in canonical form

$$\bigvee_{(P,N)\in\Pi}((\bigwedge_{s\times t\in P} s\times t)\wedge(\bigwedge_{s\times t\in N} \neg(s\times t))) \quad \bigvee_{(P,N)\in\Sigma}((\bigwedge_{s\to t\in P} s\to t)\wedge(\bigwedge_{s\to t\in N} \neg(s\to t)))$$

3. Decide (coinductively) whether the two summands are both empty by applying the ugly formulae of the previous slide.

STEP 3: Subtyping algorithm

$$s \leq t?$$

Recall that:

$$s \leq t \iff [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \iff [\![s\wedge\neg t]\!] = \varnothing \iff s\wedge\neg t = \mathbb{0}$$

1. Consider $s\wedge\neg t$
2. Put it in canonical form

$$\bigvee_{(P,N)\in\Pi}((\bigwedge_{s\times t\in P} s\times t)\wedge(\bigwedge_{s\times t\in N}\neg(s\times t))) \quad \bigvee_{(P,N)\in\Sigma}((\bigwedge_{s\to t\in P} s\to t)\wedge(\bigwedge_{s\to t\in N}\neg(s\to t)))$$

3. Decide (coinductively) whether the two summands are both empty by applying the ugly formulae of the previous slide.

STEP 3: Subtyping algorithm

$$s \leq t?$$

Recall that:

$$s \leq t \iff [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \iff [\![s \wedge \neg t]\!] = \varnothing \iff s \wedge \neg t = \mathbb{0}$$

1. Consider $s \wedge \neg t$
2. Put it in canonical form

$$\bigvee_{(P,N)\in\Pi} ((\bigwedge_{s\times t\in P} s\times t) \wedge (\bigwedge_{s\times t\in N} \neg(s\times t))) \bigvee_{(P,N)\in\Sigma} ((\bigwedge_{s\to t\in P} s\to t) \wedge (\bigwedge_{s\to t\in N} \neg(s\to t)))$$

3. Decide (coinductively) whether the two summands are both empty by applying the ugly formulae of the previous slide.

STEP 3: Subtyping algorithm

$$s \leq t?$$

Recall that:

$$s \leq t \iff [\![s]\!] \cap \overline{[\![t]\!]} = \varnothing \iff [\![s \wedge \neg t]\!] = \varnothing \iff s \wedge \neg t = \mathbb{0}$$

1. Consider $s \wedge \neg t$
2. Put it in canonical form

$$\bigvee_{(P,N)\in\Pi} ((\bigwedge_{s\times t\in P} s\times t) \wedge (\bigwedge_{s\times t\in N} \neg(s\times t))) \quad \bigvee_{(P,N)\in\Sigma} ((\bigwedge_{s\to t\in P} s\to t) \wedge (\bigwedge_{s\to t\in N} \neg(s\to t)))$$

3. Decide (coinductively) whether the two summands are both empty by applying the ugly formulae of the previous slide.

STEP 4: The Language

Lambda-abstractions: $\lambda^{\wedge_{i\in I}s_i\to t_i}x.e$

$$t\equiv(\textstyle\bigwedge_{i=1..n} s_i\to t_i)\wedge(\bigwedge_{j=1..m}\neg(s'_j\to t'_j))\neq\mathbb{0}$$

$$\text{(abstr)}\quad \frac{(\forall i)\;\; \Gamma, x : s_i \vdash_{\mathscr{D}} e : t_i}{\Gamma \vdash_{\mathscr{D}} \lambda^{\wedge_{i\in I}s_i\to t_i}x.e : t}$$

Overloading:

$$[\![(t_1\vee t_2)\to(s_1\wedge s_2)]\!] \subsetneq [\![(t_1\to s_1)\wedge(t_2\to s_2)]\!]$$

STEP 4: The Language

Lambda-abstractions: $\lambda^{\wedge_{i\in I} s_i \to t_i} x.e$

$$t \equiv (\textstyle\bigwedge_{i=1..n} s_i \to t_i) \wedge (\bigwedge_{j=1..m} \neg(s'_j \to t'_j)) \neq \mathbb{0}$$

$$(\text{abstr}) \quad \frac{(\forall i) \;\; \Gamma, x : s_i \vdash_{\mathscr{D}} e : t_i}{\Gamma \vdash_{\mathscr{D}} \lambda^{\wedge_{i\in I} s_i \to t_i} x.e : t}$$

Overloading:

$$[\![(t_1 \vee t_2) \to (s_1 \wedge s_2)]\!] \subsetneq [\![(t_1 \to s_1) \wedge (t_2 \to s_2)]\!]$$

STEP 4: The Language

Lambda-abstractions: $\lambda^{\wedge_{i\in I} s_i \to t_i} x.e$

$$t \equiv (\textstyle\bigwedge_{i=1..n} s_i \to t_i) \wedge (\bigwedge_{j=1..m} \neg(s'_j \to t'_j)) \neq \mathbb{0}$$

$$(\text{abstr}) \quad \frac{(\forall i)\quad \Gamma, x : s_i \vdash_{\mathscr{D}} e : t_i}{\Gamma \vdash_{\mathscr{D}} \lambda^{\wedge_{i\in I} s_i \to t_i} x.e : t}$$

Overloading:

$$[\![(t_1 \vee t_2) \to (s_1 \wedge s_2)]\!] \subsetneq [\![(t_1 \to s_1) \wedge (t_2 \to s_2)]\!]$$

STEP 4: The Language

Lambda-abstractions: $\lambda^{\wedge_{i\in I} s_i \to t_i} x.e$

$$t \equiv (\bigwedge_{i=1..n} s_i \to t_i) \wedge (\bigwedge_{j=1..m} \neg(s'_j \to t'_j)) \neq \mathbb{0}$$

$$(\text{abstr}) \quad \frac{(\forall i)\ \ \Gamma, x : s_i \vdash_{\mathscr{D}} e : t_i}{\Gamma \vdash_{\mathscr{D}} \lambda^{\wedge_{i\in I} s_i \to t_i} x.e : t}$$

Overloading:

$$[\![(t_1 \vee t_2) \to (s_1 \wedge s_2)]\!] \subsetneq [\![(t_1 \to s_1) \wedge (t_2 \to s_2)]\!]$$

STEP 4: The Language

Lambda-abstractions: $\lambda^{\wedge_{i\in I} s_i \to t_i} x.e$

$$t \equiv (\textstyle\bigwedge_{i=1..n} s_i \to t_i) \wedge (\bigwedge_{j=1..m} \neg(s'_j \to t'_j)) \neq \mathbb{0}$$

$$(\text{abstr}) \quad \frac{(\forall i) \quad \Gamma, x : s_i \vdash_{\mathscr{D}} e : t_i}{\Gamma \vdash_{\mathscr{D}} \lambda^{\wedge_{i\in I} s_i \to t_i} x.e : t}$$

Overloading:

$$[\![(t_1 \vee t_2) \to (s_1 \wedge s_2)]\!] \subsetneq [\![(t_1 \to s_1) \wedge (t_2 \to s_2)]\!]$$

STEP 5: Close the circle

Let $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$, then:

$$s \leq_{\mathscr{D}} t \qquad \Longleftrightarrow \qquad s \leq_{\mathscr{V}} t \tag{1}$$

Equation (1) (actually, $\Rightarrow$) states that the language is quite rich, since there always exists a value to separate two distinct types; i.e. its set of values is a model of types with "enough points"

$$s \not\leq_{\mathscr{D}} t \Longrightarrow \text{there exists } v \text{ such that } \vdash v : s \text{ and } \not\vdash v : t$$

In particular, thanks to negative arrows in (abstr) rule, the following two types:

$$\bigwedge_{i=1..k} s_i \rightarrow t_i \not\leq t$$

are distinguished by $\lambda^{\wedge_{i=1..k} s_i \rightarrow t_i} x.e$ which inhabits their difference.

For practice try the ℂDuce programming language (www.cduce.org)

STEP 5: Close the circle

Let $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$, then:

$$s \leq_{\mathscr{D}} t \qquad \Longleftrightarrow \qquad s \leq_{\mathscr{V}} t \tag{1}$$

Equation (1) (actually, $\Rightarrow$) states that the language is quite rich, since there always exists a value to separate two distinct types; i.e. its set of values is a model of types with "enough points"

$$s \not\leq_{\mathscr{D}} t \Longrightarrow \text{there exists } v \text{ such that} \vdash v : s \text{ and} \not\vdash v : t$$

In particular, thanks to negative arrows in (abstr) rule, the following two types:

$$\bigwedge_{i=1..k} s_i \to t_i \not\leq t$$

are distinguished by $\lambda^{\wedge_{i=1..k} s_i \to t_i} x.e$ which inhabits their difference.

For practice try the ℂDuce programming language (www.cduce.org)

STEP 5: Close the circle

Let $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$, then:

$$s \leq_{\mathscr{D}} t \qquad \iff \qquad s \leq_{\mathscr{V}} t \tag{1}$$

Equation (1) (actually, $\Rightarrow$) states that the language is quite rich, since there always exists a value to separate two distinct types; i.e. its set of values is a model of types with "enough points"

$$s \not\leq_{\mathscr{D}} t \Longrightarrow \text{there exists } v \text{ such that} \vdash v : s \text{ and } \not\vdash v : t$$

In particular, thanks to negative arrows in (abstr) rule, the following two types:

$$\bigwedge_{i=1..k} s_i {\rightarrow} t_i \not\leq t$$

are distinguished by $\lambda^{\wedge_{i=1..k} s_i \rightarrow t_i} x.e$ which inhabits their difference.

For practice try the ℂDuce programming language (www.cduce.org)

STEP 5: Close the circle

Let $[\![t]\!]_{\mathscr{V}} = \{v \in \mathscr{V} \mid \vdash_{\mathscr{D}} v : t\}$, then:

$$s \leq_{\mathscr{D}} t \qquad \Longleftrightarrow \qquad s \leq_{\mathscr{V}} t \tag{1}$$

Equation (1) (actually, $\Rightarrow$) states that the language is quite rich, since there always exists a value to separate two distinct types; i.e. its set of values is a model of types with "enough points"

$$s \not\leq_{\mathscr{D}} t \Longrightarrow \text{there exists } v \text{ such that } \vdash v : s \text{ and } \not\vdash v : t$$

In particular, thanks to negative arrows in (abstr) rule, the following two types:

$$\bigwedge_{i=1..k} s_i \to t_i \not\leq t$$

are distinguished by $\lambda^{\wedge_{i=1..k} s_i \to t_i} x.e$ which inhabits their difference.

For practice try the $\mathbb{C}$Duce programming language (www.cduce.org)

π-calculus.

STEP 1: types for π

Types	t	::=	b	basic types
		|	$ch^{+}(t)$	input channel type
		|	$ch^{-}(t)$	output channel type
		|	$ch(t)$	I/O channel type
		|	$\mathbb{0}$	boolean combinators
		|	$\mathbb{1}$	
		|	$\neg t$	
		|	$t \vee t$	
		|	$t \wedge t$	

STEP 1: types for π

Types	t	$::=$	b	basic types
		$\mid$	$ch^{+}(t)$	input channel type
		$\mid$	$ch^{-}(t)$	output channel type
		$\mid$	$ch(t)$	I/O channel type
		$\mid$	$\mathbb{0}$	boolean combinators
		$\mid$	$\mathbb{1}$	
		$\mid$	$\neg t$	
		$\mid$	$t \vee t$	
		$\mid$	$t \wedge t$	

STEP 2: set-theoretic model

- A channel is like a box with a particular shape
  The box can contain only objects that fit that shape
- A type t denotes the set of objects of type t.
- $[\![ch(t)]\!] = \{$channels whose shape fits objects of type $t\}$

Channels are identified by the objects they transport

STEP 2: set-theoretic model

- A channel is like a box with a particular shape
  The box can contain only objects that fit that shape
- A type t denotes the set of objects of type t.
- $[\![ch(t)]\!] = \{$channels whose shape fits objects of type $t\}$

Channels are identified by the objects they transport

- Channels are their shape

STEP 2: set-theoretic model

- A channel is like a box with a particular shape
  The box can contain only objects that fit that shape
- A type t denotes the set of objects of type t.
- $[\![ch(t)]\!] = \{$channels whose shape fits objects of type $t\}$

Channels are identified by the objects they transport

- Channels are the set of objects that fit their shape

STEP 2: set-theoretic model

- A channel is like a box with a particular shape
  The box can contain only objects that fit that shape
- A type t denotes the set of objects of type t.
- $[\![ch(t)]\!] = \{$channels whose shape fits objects of type $t\}$

Channels are identified by the objects they transport

- $[\![ch(t)]\!] = \{$set of objects of type $t\}$

STEP 2: set-theoretic model

- A channel is like a box with a particular shape
  The box can contain only objects that fit that shape
- A type t denotes the set of objects of type t.
- $[\![ch(t)]\!] = \{$channels whose shape fits objects of type $t\}$

Channels are identified by the objects they transport

- $[\![ch(t)]\!] = \{[\![t]\!]\}$

Invariance of channel types

STEP 2: set theoretic model

$ch^{+}(t)$ types all channels on which I expect to read a t-message

- $[\![ch^{+}(t)]\!] = \{[\![t']\!] \mid t' \leq t\}$

Covariance of input types

$ch^{-}(t)$ types all channels on which I'm allowed to write a t-message

- $[\![ch^{-}(t)]\!] = \{[\![t']\!] \mid t' \geq t\}$

Contravariance of output types

STEP 2: set theoretic model

$ch^{+}(t)$ types all channels on which I expect to read a t-message

- $[\![ch^{+}(t)]\!] = \{[\![t']\!] \mid t' \leq t\}$

Covariance of input types

$ch^{-}(t)$ types all channels on which I'm allowed to write a t-message

- $[\![ch^{-}(t)]\!] = \{[\![t']\!] \mid t' \geq t\}$

Contravariance of output types

STEP 2: set theoretic model

$ch^{+}(t)$ types all channels on which I expect to read a t-message

- $[\![ch^{+}(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$

Covariance of input types

$ch^{-}(t)$ types all channels on which I'm allowed to write a t-message

- $[\![ch^{-}(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

Contravariance of output types

STEP 2: set-theoretic model

Define $[\![-]\!] : \mathit{Types} \to \mathscr{P}(\mathscr{D})$:

- $[\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, $[\![t_1 \wedge t_2]\!] = [\![t_1]\!] \cap [\![t_2]\!]$, ...
- $[\![ch^{+}(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$
- $[\![ch^{-}(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

Some induced equations:

$$ch^{-}(t) \wedge ch^{+}(t) = ch(t)$$

$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t)$$

$$ch^{-}(s) \vee ch^{-}(t) \leq ch^{-}(s \wedge t)$$

Can be checked graphically

STEP 2: set-theoretic model

Define $[\![-]\!] : \mathit{Types} \to \mathscr{P}(\mathscr{D})$:

- $[\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, $[\![t_1 \wedge t_2]\!] = [\![t_1]\!] \cap [\![t_2]\!]$, ...
- $[\![ch^{+}(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$
- $[\![ch^{-}(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

We need that $\mathscr{D} = \mathbb{B} + [\![\mathscr{D}]\!]$ (not straightforward)

Some induced equations:

$$ch^{-}(t) \wedge ch^{+}(t) = ch(t)$$

$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t)$$

$$ch^{-}(s) \vee ch^{-}(t) \leq ch^{-}(s \wedge t)$$

Can be checked graphically

STEP 2: set-theoretic model

Define $[\![-]\!] : Types \to \mathscr{P}(\mathscr{D})$:

- $[\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, $[\![t_1 \wedge t_2]\!] = [\![t_1]\!] \cap [\![t_2]\!]$, ...
- $[\![ch^+(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$
- $[\![ch^-(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

Then define

$$t \leq_{\mathscr{D}} t' \iff [\![t]\!] \subseteq [\![t']\!]$$

Some induced equations:

$$ch^-(t) \wedge ch^+(t) = ch(t)$$

$$ch^-(s) \wedge ch^-(t) = ch^-(s \vee t)$$

$$ch^-(s) \vee ch^-(t) \leq ch^-(s \wedge t)$$

Can be checked graphically

STEP 2: set-theoretic model

Define $[\![-]\!] : \mathit{Types} \to \mathscr{P}(\mathscr{D})$:

- $[\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, $[\![t_1 \wedge t_2]\!] = [\![t_1]\!] \cap [\![t_2]\!]$, ...
- $[\![ch^+(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$
- $[\![ch^-(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

Then define

$$t \leq_{\mathscr{D}} t' \iff [\![t]\!] \subseteq [\![t']\!]$$

Some induced equations:

$$ch^-(t) \wedge ch^+(t) = ch(t)$$

$$ch^-(s) \wedge ch^-(t) = ch^-(s \vee t)$$

$$ch^-(s) \vee ch^-(t) \leq ch^-(s \wedge t)$$

Can be checked graphically

STEP 2: set-theoretic model

Define $[\![-]\!] : \mathit{Types} \to \mathscr{P}(\mathscr{D})$:

- $[\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, $[\![t_1 \wedge t_2]\!] = [\![t_1]\!] \cap [\![t_2]\!]$, ...
- $[\![ch^+(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$
- $[\![ch^-(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

Then define

$$t \leq_{\mathscr{D}} t' \iff [\![t]\!] \subseteq [\![t']\!]$$

Some induced equations:

$$\boxed{ch^-(t) \wedge ch^+(t) = ch(t)}$$

$$ch^-(s) \wedge ch^-(t) = ch^-(s \vee t)$$

$$ch^-(s) \vee ch^-(t) \leq ch^-(s \wedge t)$$

Can be checked graphically

STEP 2: set-theoretic model

Define $[\![-]\!] : \mathit{Types} \to \mathscr{P}(\mathscr{D})$:

- $[\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!]$, $[\![t_1 \wedge t_2]\!] = [\![t_1]\!] \cap [\![t_2]\!]$, ...
- $[\![ch^+(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \subseteq [\![t]\!]\}$
- $[\![ch^-(t)]\!] = \{[\![t']\!] \mid [\![t']\!] \supseteq [\![t]\!]\}$

Then define

$$t \leq_{\mathscr{D}} t' \iff [\![t]\!] \subseteq [\![t']\!]$$

Some induced equations:

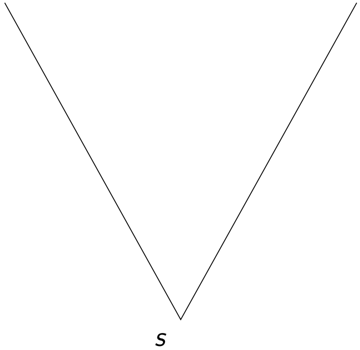
$$ch^-(t) \wedge ch^+(t) = ch(t)$$

$$ch^-(s) \wedge ch^-(t) = ch^-(s \vee t)$$

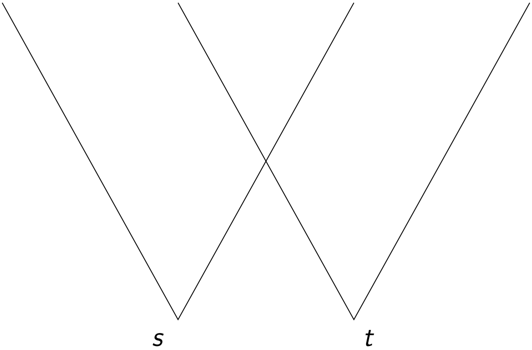
$$ch^-(s) \vee ch^-(t) \leq ch^-(s \wedge t)$$

Can be checked graphically

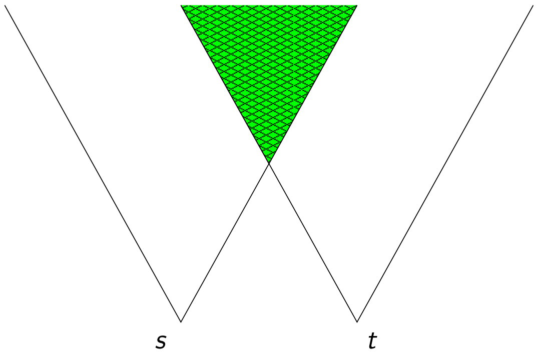
$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t)$$



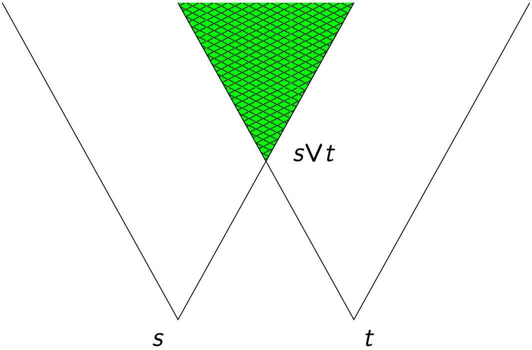
$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t)$$



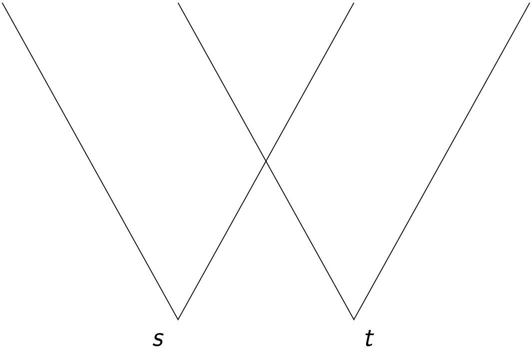
$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t)$$



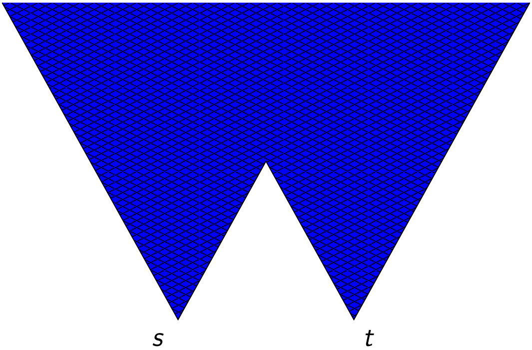
$$ch^{-}(s) \wedge ch^{-}(t) = ch^{-}(s \vee t)$$



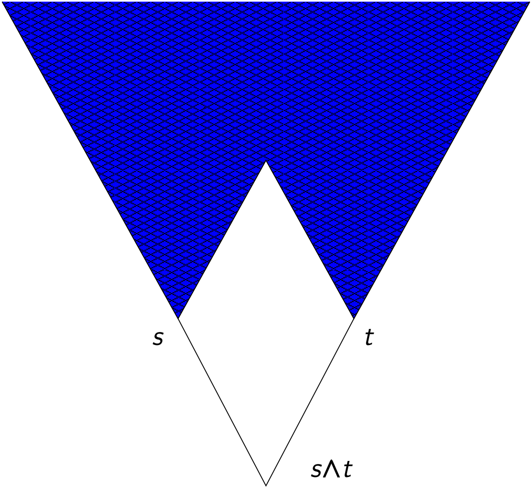
$$ch^{-}(s) \vee ch^{-}(t) \not\leq ch^{-}(s \wedge t)$$



$$ch^{-}(s) \vee ch^{-}(t) \not\leq ch^{-}(s \wedge t)$$



$$ch^{-}(s) \vee ch^{-}(t) \not\leq ch^{-}(s \wedge t)$$



STEP 3: Subtyping Algorithm

It is enough to decide emptiness:

$$s \leq t \iff s \wedge \neg t = \mathbb{0}$$

Put $s \wedge \neg t$ in disjunctive normal form
A disjunction is empty is all the summands are empty

$$\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbb{0}?$$

Equivalently

STEP 3: Subtyping Algorithm

It is enough to decide emptiness:

$$s \leq t \iff s\wedge\neg t = \mathbb{0}$$

Put $s\wedge\neg t$ in disjunctive normal form
A disjunction is empty is all the summands are empty

$$\bigwedge_{i\in P} t_i \wedge \bigwedge_{j\in N} \neg t'_j = \mathbb{0}?$$

Equivalently

STEP 3: Subtyping Algorithm

It is enough to decide emptiness:

$$s \leq t \quad \Longleftrightarrow \quad s \wedge \neg t = \mathbb{0}$$

Put $s \wedge \neg t$ in disjunctive normal form
A disjunction is empty is all the summands are empty

$$\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbb{0}?$$

Equivalently

STEP 3: Subtyping Algorithm

It is enough to decide emptiness:

$$s \leq t \quad \Longleftrightarrow \quad s \wedge \neg t = \mathbb{0}$$

Put $s \wedge \neg t$ in disjunctive normal form
A disjunction is empty is all the summands are empty

$$\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbb{0}?$$

Equivalently

$$\bigwedge_{i \in P} t_i \leq \bigvee_{j \in N} t'_j \ ?$$

STEP 3: Subtyping Algorithm

It is enough to decide emptiness:

$$s \leq t \quad \Longleftrightarrow \quad s \wedge \neg t = \mathbb{0}$$

Put $s \wedge \neg t$ in disjunctive normal form
A disjunction is empty is all the summands are empty

$$\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbb{0}?$$

Equivalently

$$\bigwedge_{i \in I} ch^{+}(t_1^i) \wedge \bigwedge_{j \in J} ch^{-}(t_2^j) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$

STEP 3: Subtyping Algorithm

It is enough to decide emptiness:

$$s \leq t \quad \Longleftrightarrow \quad s \wedge \neg t = \mathbb{0}$$

Put $s \wedge \neg t$ in disjunctive normal form

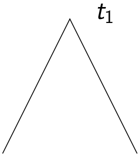
A disjunction is empty is all the summands are empty

$$\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbb{0}?$$

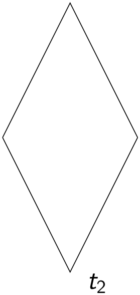
Equivalently

$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$

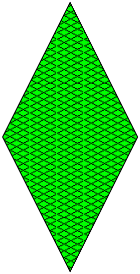
$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



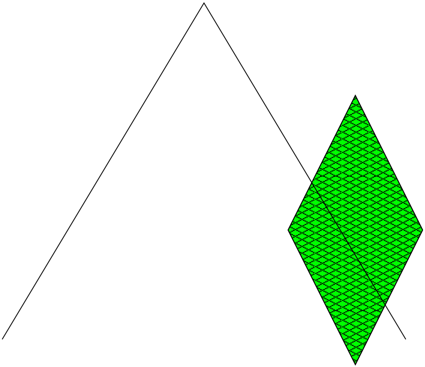
$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



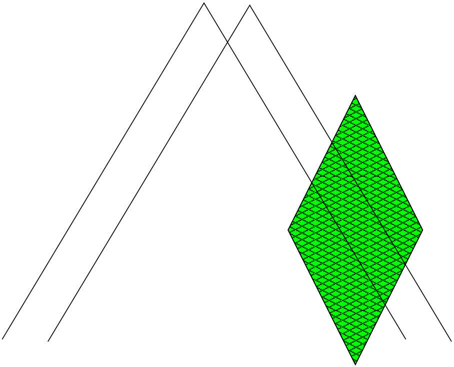
$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



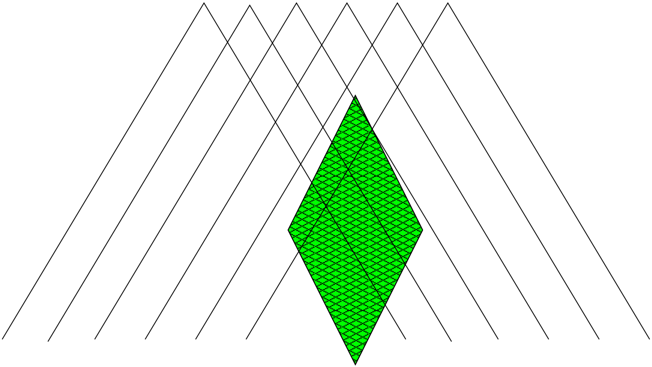
$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



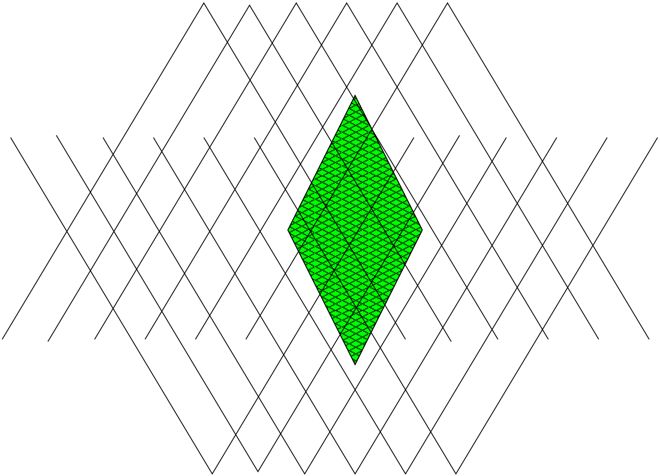
$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^h) \vee \bigvee_{k \in K} ch^{-}(t_4^k)$$



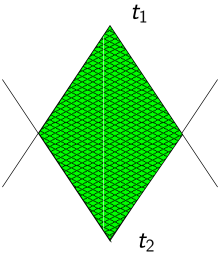
Atoms

In some cases the condition to check involves atoms
Types with a singleton interpretation

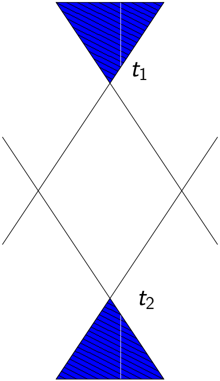
Consider:

- two types $t_1 \neq t_2$
- $t_2 \leq t_1$
- question: is $ch^+(t_1) \wedge ch^-(t_2) \leq ch^+(t_2) \vee ch^-(t_1)$?

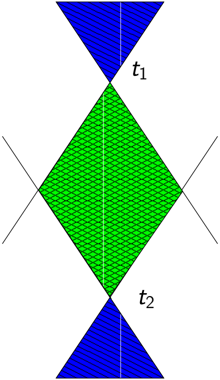
$ch^{+}(t_1) \wedge ch^{-}(t_2)$ $ch^{+}(t_2) \vee ch^{-}(t_1)$



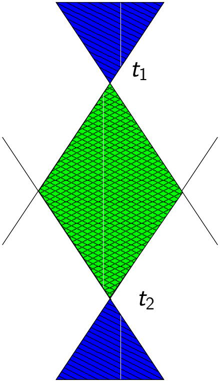
$ch^{+}(t_1) \wedge ch^{-}(t_2)$ $\quad$ $ch^{+}(t_2) \vee ch^{-}(t_1)$



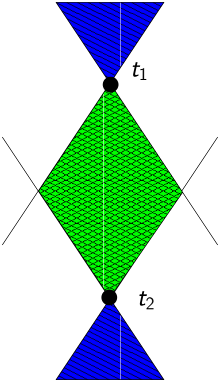
$ch^{+}(t_1) \wedge ch^{-}(t_2)$ $ch^{+}(t_2) \vee ch^{-}(t_1)$



$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq ch^{+}(t_2) \vee ch^{-}(t_1)$$

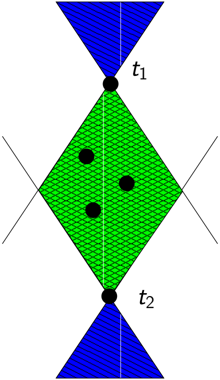


$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq ch^{+}(t_2) \vee ch^{-}(t_1)$$



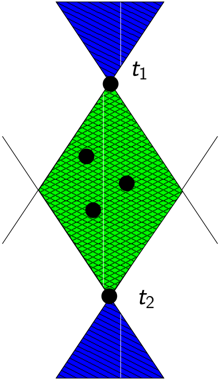
The two sets have these two points in common, namely $ch(t_1)$ and $ch(t_2)$.

$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq ch^{+}(t_2) \vee ch^{-}(t_1)$$



The disequation holds if there are no other points in the left hand side

$$ch^{+}(t_1) \wedge ch^{-}(t_2) \leq ch^{+}(t_2) \vee ch^{-}(t_1)$$



It depends on whether $t_1 \wedge \neg t_2$ is atomic: that is whether there is nothing between t_1 and t_2

STEP 4: The Language

Which kind of π-calculus?

Consider again

$$ch^{+}(\mathtt{t}_1) \vee ch^{+}(\mathtt{t}_2) \lneq ch^{+}(\mathtt{t}_1 \vee \mathtt{t}_2)$$

- Containment is strict, so we want programs that distinguish these two types
- We must be able to dynamically check the type of messages arriving a channel
- Use type-case in read actions.

STEP 4: The Language

Which kind of π-calculus?

Consider again

$$ch^{+}(\mathtt{t}_1) \vee ch^{+}(\mathtt{t}_2) \lneqq ch^{+}(\mathtt{t}_1 \vee \mathtt{t}_2)$$

- Containment is strict, so we want programs that distinguish these two types
- We must be able to dynamically check the type of messages arriving a channel
- Use type-case in read actions.

STEP 4: The Language

Which kind of π-calculus?

Consider again

$$ch^{+}(\mathtt{t}_1) \vee ch^{+}(\mathtt{t}_2) \lneqq ch^{+}(\mathtt{t}_1 \vee \mathtt{t}_2)$$

- Containment is strict, so we want programs that distinguish these two types
- We must be able to dynamically check the type of messages arriving a channel
- Use type-case in read actions.

STEP 4: The Language

Which kind of π-calculus?

Consider again

$$ch^{+}(\mathtt{t}_1) \vee ch^{+}(\mathtt{t}_2) \lneq ch^{+}(\mathtt{t}_1 \vee \mathtt{t}_2)$$

- Containment is strict, so we want programs that distinguish these two types
- We must be able to dynamically check the type of messages arriving a channel
- Use type-case in read actions.

STEP 4: The Language

Which kind of π-calculus?

Consider again

$$ch^{+}(\mathtt{t}_1)\vee ch^{+}(\mathtt{t}_2) \lneq ch^{+}(\mathtt{t}_1\vee\mathtt{t}_2)$$

- Containment is strict, so we want programs that distinguish these two types
- We must be able to dynamically check the type of messages arriving a channel
- Use type-case in read actions.

STEP 4: The Language — $\mathbb{C}\pi$-calculus

Channels	α	$::=$	x	variables
		$\mid$	c^t	channel constants
Processes	P	$::=$	$\overline{\alpha}(\alpha)$	output
		$\mid$	$\sum_{i\in I}\alpha(x{:}t_i).P_i$	guarded input
		$\mid$	$P_1 \| P_2$	parallel
		$\mid$	$(\nu c^t)P$	restriction
		$\mid$	$!P$	replication

Reduction

$$\overline{c}_1^s(c_2^t) \parallel \sum_{i\in I} c_1^s(x:t_i)P_i \quad\rightarrow\quad P_j[c_2^t/x] \qquad \text{if } ch(t) \leq t_j$$

STEP 4: The Language $\mathbb{C}\pi$-calculus

Channels	α	$::=$	x	variables
		$\mid$	c^t	channel constants
Processes	P	$::=$	$\overline{\alpha}(\alpha)$	output
		$\mid$	$\sum_{i\in I} \alpha(x{:}t_i).P_i$	guarded input
		$\mid$	$P_1 \| P_2$	parallel
		$\mid$	$(\nu c^t)P$	restriction
		$\mid$	$!P$	replication

Reduction

$$\overline{c}_1^s(c_2^t) \parallel \sum_{i\in I} c_1^s(x : t_i)P_i \quad \rightarrow \quad P_j[c_2^t/x] \qquad \text{if } ch(t) \leq t_j$$

STEP 4: The Language $\mathbb{C}\pi$-calculus

Channels	α	$::=$	x	variables
		$\mid$	c^t	channel constants
Processes	P	$::=$	$\overline{\alpha}(\alpha)$	output
		$\mid$	$\sum_{i\in I}\alpha(x{:}t_i).P_i$	guarded input
		$\mid$	$P_1 \| P_2$	parallel
		$\mid$	$(\nu c^t)P$	restriction
		$\mid$	$!P$	replication

Reduction

$$\overline{c}_1^s(c_2^t) \parallel \sum_{i\in I} c_1^s(x : t_i)P_i \quad \rightarrow \quad P_j[c_2^t/x] \qquad \text{if } ch(t) \leq t_j$$

Type case

STEP 4: The Language $\mathbb{C}\pi$-calculus

Channels	α	$::=$	x	variables
		$\mid$	c^t	channel constants
Processes	P	$::=$	$\overline{\alpha}(\alpha)$	output
		$\mid$	$\sum_{i\in I} \alpha(x{:}t_i).P_i$	guarded input
		$\mid$	$P_1 \| P_2$	parallel
		$\mid$	$(\nu c^t)P$	restriction
		$\mid$	$!P$	replication

Reduction

$$\overline{c}_1^s(c_2^t) \parallel \sum_{i\in I} c_1^s(x : t_i)P_i \quad \rightarrow \quad P_j[c_2^t/x] \qquad \text{if } ch(t) \leq t_j$$

Type case

STEP 4: The Language $\mathbb{C}\pi$-calculus

Channels	α	$::=$	x	variables
		$\mid$	c^t	channel constants
Processes	P	$::=$	$\overline{\alpha}(\alpha)$	output
		$\mid$	$\sum_{i\in I} \alpha(x{:}t_i).P_i$	guarded input
		$\mid$	$P_1 \Vert P_2$	parallel
		$\mid$	$(\nu c^t)P$	restriction
		$\mid$	$!P$	replication

Reduction

$$\overline{c}_1^s(\boldsymbol{c_2^t}) \parallel \sum_{i\in I} c_1^s(x : t_i)P_i \quad \rightarrow \quad P_j[c_2^t/x] \qquad \text{if } ch(t) \leq t_j$$

Type case call-by-value

STEP 4: The Language $\mathbb{C}\pi$-calculus

Channels	α	$::=$	x	variables
		$\mid$	c^t	channel constants
Processes	P	$::=$	$\overline{\alpha}(\alpha)$	output
		$\mid$	$\sum_{i\in I} \alpha(x{:}t_i).P_i$	guarded input
		$\mid$	$P_1 \Vert P_2$	parallel
		$\mid$	$(\nu c^t)P$	restriction
		$\mid$	$!P$	replication

Reduction

$$\overline{c}_1^s(c_2^t) \parallel \sum_{i\in I} c_1^s(x : t_i)P_i \quad \to \quad P_j[c_2^t/x] \qquad \text{if } ch(t) \leq t_j$$

Type case call-by-value

STEP 4: The Language (Typing rules)

$$\frac{}{\Gamma \vdash c^t : ch(t)}\ (\text{chan}) \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{var}) \qquad \frac{\Gamma \vdash \alpha : s \leq_{\mathscr{D}} t}{\Gamma \vdash \alpha : t}\ (\text{subsum})$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash (\nu c^t)P}\ (\text{new}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}\ (\text{repl}) \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \| P_2}\ (\text{para})$$

Matching is exhaustive

$$\begin{array}{c} t \leq \bigvee_{i\in I} t_i \\ t_i \wedge t \neq \mathbb{0} \end{array} \quad \frac{\Gamma \vdash \alpha : ch^{+}(t) \qquad \Gamma, x{:}t_i \vdash P_i}{\Gamma \vdash \sum_{i\in I} \alpha(x{:}t_i).P_i}\ (\text{input})$$

No useless branch

$$\frac{\Gamma \vdash \beta : t \qquad \Gamma \vdash \alpha : ch^{-}(t)}{\Gamma \vdash \overline{\alpha}(\beta)}\ (\text{output})$$

STEP 4: The Language (Typing rules)

$$\frac{}{\Gamma \vdash c^t : ch(t)}\ (\text{chan}) \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{var}) \qquad \frac{\Gamma \vdash \alpha : s \leq_{\mathscr{D}} t}{\Gamma \vdash \alpha : t}\ (\text{subsum})$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash (\nu c^t)P}\ (\text{new}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}\ (\text{repl}) \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \| P_2}\ (\text{para})$$

Matching is exhaustive

$$\begin{array}{c} t \leq \bigvee_{i\in I} t_i \\ t_i \wedge t \neq \mathbb{0} \end{array} \quad \frac{\Gamma \vdash \alpha : ch^+(t) \qquad \Gamma, x{:}t_i \vdash P_i}{\Gamma \vdash \sum_{i\in I} \alpha(x{:}t_i).P_i}\ (\text{input})$$

No useless branch

$$\frac{\Gamma \vdash \beta : t \qquad \Gamma \vdash \alpha : ch^-(t)}{\Gamma \vdash \overline{\alpha}(\beta)}\ (\text{output})$$

STEP 4: The Language (Typing rules)

$$\frac{}{\Gamma \vdash c^t : ch(t)}\ (\text{chan}) \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{var}) \qquad \frac{\Gamma \vdash \alpha : s \leq_{\mathscr{D}} t}{\Gamma \vdash \alpha : t}\ (\text{subsum})$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash (\nu c^t)P}\ (\text{new}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}\ (\text{repl}) \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \| P_2}\ (\text{para})$$

Matching is exhaustive

$$\begin{array}{c} t \leq \bigvee_{i \in I} t_i \\ t_i \wedge t \neq \mathbb{0} \end{array} \quad \frac{\Gamma \vdash \alpha : ch^+(t) \qquad \Gamma, x{:}t_i \vdash P_i}{\Gamma \vdash \sum_{i \in I} \alpha(x{:}t_i).P_i}\ (\text{input})$$

No useless branch

$$\frac{\Gamma \vdash \beta : t \qquad \Gamma \vdash \alpha : ch^-(t)}{\Gamma \vdash \overline{\alpha}(\beta)}\ (\text{output})$$

STEP 4: The Language (Typing rules)

$$\frac{}{\Gamma \vdash c^t : ch(t)}\ (\text{chan}) \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{var}) \qquad \frac{\Gamma \vdash \alpha : s \leq_{\mathscr{D}} t}{\Gamma \vdash \alpha : t}\ (\text{subsum})$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash (\nu c^t)P}\ (\text{new}) \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}\ (\text{repl}) \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \| P_2}\ (\text{para})$$

Matching is exhaustive

$$t \leq \bigvee_{i\in I} t_i \qquad t_i \wedge t \neq \mathbb{0} \qquad \frac{\Gamma \vdash \alpha : ch^+(t) \qquad \Gamma, x{:}t_i \vdash P_i}{\Gamma \vdash \sum_{i\in I} \alpha(x{:}t_i).P_i}\ (\text{input})$$

No useless branch

$$\frac{\Gamma \vdash \beta : t \qquad \Gamma \vdash \alpha : ch^-(t)}{\Gamma \vdash \overline{\alpha}(\beta)}\ (\text{output})$$

STEP 5: Closing the circle

As usual for

$$[\![t]\!]_{\mathscr{V}} = \{v \mid\ \vdash v : t\}$$

One has

$$s \leq_{\mathscr{D}} t \iff s \leq_{\mathscr{V}} t$$

Note that we did not use negated types in inference rules

STEP 5: Closing the circle

As usual for

$$[\![t]\!]_{\mathscr{V}} = \{v \mid \vdash v : t\}$$

One has

$$s \leq_{\mathscr{D}} t \quad \iff \quad s \leq_{\mathscr{V}} t$$

Note that we did not use negated types in inference rules

Some Perspectives.

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \overset{\text{def}}{\Longleftrightarrow} \quad \forall s. [\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Atomic types

In $\mathbb{C}\pi$ subtyping check requires atomicity check

The same happens in λ-calculus as soon as we extend it by polymorphic types:

$$t_1 \leq t_2 \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall s.[\![t_1[s/X]]\!] \subseteq [\![t_2[s/X]]\!]$$

Consider

$$(t \times X) \leq (t \times \neg t) \vee (X \times t)$$

holds true iff t is atomic (ie, for all X, either $t \leq X$ or $t \leq \neg X$)

Relation between atomicity and semantic subtyping?

Consider systems where atomic types are not denotable?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \mathtt{int} \vee (ch(\mathtt{int}) \wedge ch(t))$$

Is t equal to $\mathtt{int}$?

$$t = \mathtt{int} \quad \Rightarrow \quad (ch(\mathtt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \mathtt{int}$$

$$t \neq \mathtt{int} \quad \Rightarrow \quad (ch(\mathtt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \mathtt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \texttt{int} \vee (ch(\texttt{int}) \wedge ch(t))$$

Is t equal to $\texttt{int}$?

$$t = \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \texttt{int}$$

$$t \neq \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \texttt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \texttt{int} \vee (ch(\texttt{int}) \wedge ch(t))$$

Is t equal to $\texttt{int}$?

$$t = \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \texttt{int}$$

$$t \neq \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \texttt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \texttt{int} \vee (ch(\texttt{int}) \wedge ch(t))$$

Is t equal to $\texttt{int}$?

$$t = \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \texttt{int}$$

$$t \neq \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \texttt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \mathtt{int} \vee (ch(\mathtt{int}) \wedge ch(t))$$

Is t equal to $\mathtt{int}$?

$$t = \mathtt{int} \quad \Rightarrow \quad (ch(\mathtt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \mathtt{int}$$

$$t \neq \mathtt{int} \quad \Rightarrow \quad (ch(\mathtt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \mathtt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \texttt{int} \vee (ch(\texttt{int}) \wedge ch(t))$$

Is t equal to `int`?

$$t = \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \texttt{int}$$

$$t \neq \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \texttt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \mathtt{int} \vee (ch(\mathtt{int}) \wedge ch(t))$$

Is t equal to $\mathtt{int}$?

$$t = \mathtt{int} \quad \Rightarrow \quad (ch(\mathtt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \mathtt{int}$$

$$t \neq \mathtt{int} \quad \Rightarrow \quad (ch(\mathtt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \mathtt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Recursive types and models

In some cases, a set-theoretic model does not exists:

$$t = \texttt{int} \vee (ch(\texttt{int}) \wedge ch(t))$$

Is t equal to $\texttt{int}$?

$$t = \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) = \varnothing \quad \Rightarrow \quad t \neq \texttt{int}$$

$$t \neq \texttt{int} \quad \Rightarrow \quad (ch(\texttt{int}) \wedge ch(t)) \neq \varnothing \quad \Rightarrow \quad t = \texttt{int}$$

Same problem in λ-calculus when we add reference types

What the non-existence of a set-theoretic model means?

Mathematical limit or approach's limit?

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid ch^{+}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} \alpha(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid\ !P
\end{array}
$$

- known to be expressive enough
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid ch^{+}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t\vee t \mid t\wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i\in I} \alpha(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid\ !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid ch^{+}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} \alpha(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid ch^{+}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} \alpha\,(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid\ !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{rcl}
t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} \alpha(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i\in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$\begin{array}{lcl} t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\ \alpha & ::= & x \mid c^{t} \\ P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P \end{array}$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^+()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid \cancel{ch^+(t)} \mid ch^-(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^t \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i\in I} c^t(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^t)P \mid\ !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid \,!P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

$$
ch^{+}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h \in H} ch^{+}(t_3^{h}) \vee \bigvee_{k \in K} ch^{-}(t_4^{k})
$$

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i\in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

$$
\cancel{ch^{+}}(t_1) \wedge ch^{-}(t_2) \leq \bigvee_{h\in H} \cancel{ch^{+}}(t_3^h) \vee \bigvee_{k\in K} ch^{-}(t_4^k)
$$

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$\begin{array}{lcl} t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\ \alpha & ::= & x \mid c^{t} \\ P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i\in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid\ !P \end{array}$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

$$ch^{-}(t) \leq \bigvee_{k\in K} ch^{-}(t^{k}) \iff \exists k \in K\,.\, t^{k} \leq t$$

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$\begin{array}{lcl} t & ::= & b \mid \cancel{ch^{+}(t)} \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\ \alpha & ::= & x \mid c^{t} \\ P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P \end{array}$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

$$ch^{-}(t) \leq \bigvee_{k \in K} ch^{-}(t^{k}) \iff \exists k \in K \,.\, t^{k} \leq t$$

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i \in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

$$ch^{-}(t) \leq \bigvee_{k \in K} ch^{-}(t^{k}) \iff \exists k \in K \,.\, t^{k} \leq t$$

Unfortunately

$\not\vdash c^{\mathtt{int}} : \neg ch^{-}(\mathtt{bool})$ (here $c^{\mathtt{int}} : ch^{-}(\mathtt{int})$ instead of $ch(\mathtt{int})$)

Enough points and inference of negation

Full recursion is possible in *local* version of $\mathbb{C}\pi$.

- received channels cannot be used in input
- the type $ch^{+}()$ is not needed: we can have full recursion

$$
\begin{array}{lcl}
t & ::= & b \mid \cancel{ch^{+}}(t) \mid ch^{-}(t) \mid \mathbb{0} \mid \mathbb{1} \mid \neg t \mid t \vee t \mid t \wedge t \\
\alpha & ::= & x \mid c^{t} \\
P & ::= & \overline{\alpha}(\alpha) \mid \sum_{i\in I} c^{t}(x{:}t_i).P_i \mid P_1 \| P_2 \mid (\nu c^{t})P \mid !P
\end{array}
$$

- known to be expressive enough (encodes λ)
- decidability (even with arrow types) is straightforward

$$ch^{-}(t) \leq \bigvee_{k\in K} ch^{-}(t^{k}) \iff \exists k\in K\,.\,t^{k} \leq t$$

Unfortunately

$\not\vdash c^{\mathtt{int}} : \neg ch^{-}(\mathtt{bool})$ (here $c^{\mathtt{int}} : ch^{-}(\mathtt{int})$ instead of $ch(\mathtt{int})$)

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^-(t) \wedge \neg ch^-(t_1) \wedge \ldots \wedge \neg ch^-(t_n)} \text{(chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?
Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^-(t) \wedge \neg ch^-(t_1) \wedge \ldots \wedge \neg ch^-(t_n)} \text{ (chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?
Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^{-}(t) \wedge \neg ch^{-}(t_1) \wedge \ldots \wedge \neg ch^{-}(t_n)} \text{ (chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?
Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^-(t) \wedge \neg ch^-(t_1) \wedge \ldots \wedge \neg ch^-(t_n)} \text{ (chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?
Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^{-}(t) \wedge \neg ch^{-}(t_1) \wedge \ldots \wedge \neg ch^{-}(t_n)} \ \text{(chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?
Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^-(t) \wedge \neg ch^-(t_1) \wedge \ldots \wedge \neg ch^-(t_n)} \text{ (chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?

Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Enough points and inference of negation

Solution: use the same techniques as (abstr):

$$\frac{t_i \not\leq t}{\Gamma \vdash c^t : ch^-(t) \wedge \neg ch^-(t_1) \wedge \ldots \wedge \neg ch^-(t_n)} \text{ (chan)}$$

As for (abstr) the rule (chan) inhabits non-empty types so that values form a set-theoretic model

Such rules are problematic: no intuition and no minimum typing property

Avoidable by "schemata" but intuition is not recovered and values no longer yield a model

Inference of negations?

Have these rules a mathematical meaning?
Is it possible to find for λ a "trick" as local-$\mathbb{C}\pi$?

Some further issues

- **Polymorphic types**
  A lot of work carried on (implicit/explicit, parametricity,...) especially in the field of programming language for XML
- **Type cases**
  How much related is semantic subtyping to the presence of typecases? Not strictly necessary but make things easier (PL viewpoint) of worst (mathematical viewpoint)
- **Dependent types**
  An unexplored country.
- **Relating semantic-λ and semantic-π**
  Fascinating as it goes deep into the relation among overloading, sequentiality, and concurrency.

Main question

Is semantic subtyping just a definition technique or something more?

Some further issues

- **Polymorphic types**
  A lot of work carried on (implicit/explicit, parametricity,. . .) especially in the field of programming language for XML
- **Type cases**
  How much related is semantic subtyping to the presence of typecases? Not strictly necessary but make things easier (PL viewpoint) of worst (mathematical viewpoint)
- **Dependent types**
  An unexplored country.
- **Relating semantic-λ and semantic-π**
  Fascinating as it goes deep into the relation among overloading, sequentiality, and concurrency.

Main question

Is semantic subtyping just a definition technique or something more?

Some further issues

- **Polymorphic types**
  A lot of work carried on (implicit/explicit, parametricity,...) especially in the field of programming language for XML
- **Type cases**
  How much related is semantic subtyping to the presence of typecases? Not strictly necessary but make things easier (PL viewpoint) of worst (mathematical viewpoint)
- **Dependent types**
  An unexplored country.
- **Relating semantic-λ and semantic-π**
  Fascinating as it goes deep into the relation among overloading, sequentiality, and concurrency.

Main question

Is semantic subtyping just a definition technique or something more?

Some further issues

- **Polymorphic types**
  A lot of work carried on (implicit/explicit, parametricity,. . .) especially in the field of programming language for XML
- **Type cases**
  How much related is semantic subtyping to the presence of typecases? Not strictly necessary but make things easier (PL viewpoint) of worst (mathematical viewpoint)
- **Dependent types**
  An unexplored country.
- **Relating semantic-λ and semantic-π**
  Fascinating as it goes deep into the relation among overloading, sequentiality, and concurrency.

Main question

Is semantic subtyping just a definition technique or something more?

Some further issues

- **Polymorphic types**
  A lot of work carried on (implicit/explicit, parametricity,. . .) especially in the field of programming language for XML
- **Type cases**
  How much related is semantic subtyping to the presence of typecases? Not strictly necessary but make things easier (PL viewpoint) of worst (mathematical viewpoint)
- **Dependent types**
  An unexplored country.
- **Relating semantic-λ and semantic-π**
  Fascinating as it goes deep into the relation among overloading, sequentiality, and concurrency.

Main question

Is semantic subtyping just a definition technique or something more?

Some further issues

- **Polymorphic types**
  A lot of work carried on (implicit/explicit, parametricity,...) especially in the field of programming language for XML
- **Type cases**
  How much related is semantic subtyping to the presence of typecases? Not strictly necessary but make things easier (PL viewpoint) of worst (mathematical viewpoint)
- **Dependent types**
  An unexplored country.
- **Relating semantic-λ and semantic-π**
  Fascinating as it goes deep into the relation among overloading, sequentiality, and concurrency.

Main question

Is semantic subtyping just a definition technique or something more?

Conclusion

La morale de l'histoire est ...

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. Define a set-theoretic interpretation $[\![\ ]\!]$ for your type constructors so that it matches your semantic intuition
2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise tweak it.
3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence derive a subtyping algorithm.
4. Enjoy.

La morale de l'histoire est . . .

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. Define a set-theoretic interpretation $[\![\ ]\!]$ for your type constructors so that it matches your semantic intuition
2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise tweak it.
3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence derive a subtyping algorithm.
4. **Enjoy.**

La morale de l'histoire est ...

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. Define a set-theoretic interpretation $[\![\ ]\!]$ for your type constructors so that it matches your semantic intuition
2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise tweak it.
3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence derive a subtyping algorithm.
4. Enjoy.

La morale de l'histoire est . . .

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. Define a set-theoretic interpretation $[\![\ ]\!]$ for your type constructors so that it matches your semantic intuition

2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise tweak it.

3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence derive a subtyping algorithm.

4. Enjoy.

La morale de l'histoire est . . .

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. Define a set-theoretic interpretation $[\![\ ]\!]$ for your type constructors so that it matches your semantic intuition
2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise tweak it.
3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence derive a subtyping algorithm.
4. Enjoy.

La morale de l'histoire est . . .

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. **Define a set-theoretic interpretation** $[\![\;]\!]$ for your type constructors so that it matches your semantic intuition
   [may be not easy/possible]
2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise **tweak it**. [may be not easy/possible]
3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence **derive a subtyping algorithm**.
   [may be not easy/possible]
4. Enjoy.

La morale de l'histoire est . . .

If you have a strong semantic intuition of your favourite language and you want to add set-theoretic $\vee$, $\wedge$, $\neg$ types then:

1. **Define a set-theoretic interpretation** $[\![\ ]\!]$ for your type constructors so that it matches your semantic intuition
2. Use the subtyping relation induced by the model to type your language: if the intuition was right then the set of values is also a model, otherwise **tweak it**.
3. Use the set-theoretic properties of the model to decompose the emptiness test for your type constructors, and hence **derive a subtyping algorithm**.
4. **Enjoy.**