

# The ProcessJ Type System

Matt Bækgaard Pedersen

October 9, 2018

## Abstract

## 1 ProcessJ Types

### 1.1 Primitive Types

ProcessJ has 11 primitive types shown in Table 1.

Table 1: ProcessJ primitive types

Type Name	Values
byte	{true, false}
short	
char	
int	
long	
float	
double	
bool	
string	
barrier	
timer	

### 1.2 Constructed Types

If we count mobile procedures, ProcessJ has 6 constructed types; they are shown in Table 2.

Table 2: ProcessJ constructed types

Type Name	Representation
Array	$Array(\alpha, index)$
Record	$Record(name, \{(n_1, t_1), \dots, (n_m, t_m)\})$
Protocol	$Protocol(name, \{(tag_1, \{(n_{1,1}, t_{1,1}), \dots, (n_{1,m_1}, t_{1,m_1})\})\}$ $\vdots$ $(tag_k, \{(n_{k,1}, t_{k,1}), \dots, (n_{k,m_k}, t_{k,m_k})\})\})$
Channel	$Channel(\alpha, access)$
Channel end	$ChannelEnd(\alpha, end)$ , $\alpha$ is a Channel.
Procedure	$Procedure(name, (t_1, t_2, \dots, t_n), t)$

## 2 Helper Functions

### 2.1 Ceiling

$$\lceil \alpha, \beta \rceil := \begin{cases} \alpha & \beta \leq_{\mathcal{T}} \alpha \\ \beta & \alpha <_{\mathcal{T}} \beta \\ \perp & \text{otherwise} \end{cases}$$

### 2.2 $<_{\mathcal{T}}$

#### 2.2.1 Primitive Types

A primitive type  $\alpha$  is “type-wise less than” another primitive type  $\beta$ , if a variable of type  $\beta$  can hold any value of type  $\alpha$ . This definition seems to be exactly that of assignment compatible – and it is, but it has to be defined somewhere, so here we go:

$$\text{byte} <_{\mathcal{T}} \text{short} <_{\mathcal{T}} \text{char} <_{\mathcal{T}} \text{int} <_{\mathcal{T}} \text{long}$$

as well as

$$\text{float} <_{\mathcal{T}} \text{double}$$

but also

$$\text{int} <_{\mathcal{T}} \text{float} \wedge \text{long} <_{\mathcal{T}} \text{double}$$

$<_{\mathcal{T}}$  is, of course, transitive, so if  $(\alpha <_{\mathcal{T}} \beta) \wedge (\beta <_{\mathcal{T}} \delta) \Rightarrow \alpha <_{\mathcal{T}} \delta$ .

$\alpha <_{\mathcal{T}} \beta$	byte	short	char	int	long	float	double	bool	string	barrier	timer
byte	F	T	T	T	T	T	T	F	F	F	F
short	F	F	T	T	T	T	T	F	F	F	F
char	F	F	F	T	T	T	T	F	F	F	F
int	F	F	F	F	T	T	T	F	F	F	F
long	F	F	F	F	F	T	T	F	F	F	F
float	F	F	F	F	F	F	T	F	F	F	F
double	F	F	F	F	F	F	F	F	F	F	F
bool	F	F	F	F	F	F	F	F	F	F	F
string	F	F	F	F	F	F	F	F	F	F	F
barrier	F	F	F	F	F	F	F	F	F	F	F
timer	F	F	F	F	F	F	F	F	F	F	F

#### 2.2.2 Constructed Types

##### Protocol

$$\begin{aligned} \alpha &= \text{Protocol}(\text{name}_1, \{(tag_{1,1}, \{(n_{1,1,1}, t_{1,1,1}), \dots, (n_{1,1,m_{1,1}}, t_{1,1,m_{1,1}})\}), \\ &\quad (tag_{1,2}, \{(n_{1,2,1}, t_{1,2,1}), \dots, (n_{1,2,m_{1,2}}, t_{1,2,m_{1,2}})\}), \\ &\quad \vdots \\ &\quad (tag_{1,k_1}, \{(n_{1,k_1,1}, t_{1,k_1,1}), \dots, (n_{1,k_1,m_{1,k_1}}, t_{1,k_1,m_{1,k_1}})\})\}) \\ \beta &= \text{Protocol}(\text{name}_2, \{(tag_{2,1}, \{(n_{2,1,1}, t_{2,1,1}), \dots, (n_{2,1,m_{2,1}}, t_{2,1,m_{2,1}})\}), \\ &\quad (tag_{2,2}, \{(n_{2,2,1}, t_{2,2,1}), \dots, (n_{2,2,m_{2,2}}, t_{2,2,m_{2,2}})\}), \\ &\quad \vdots \end{aligned}$$

$$(tag_{2,k_2}, \{(n_{2,k_2,1}, t_{3,k_2,1}), \dots, (n_{2,k_2,m_{2,k_2}}, t_{1,k_2,m_{2,k_2}})\}\})$$

$$\alpha \leq_{\mathcal{T}} \beta \Leftrightarrow (\forall i : (1 \leq i \leq k_1) : \exists j : (1 \leq j \leq k_2) : tag_{1,i} = tag_{2,j} \wedge$$

$$(m_{1,i} = m_{2,j}) \wedge \bigwedge_{k=1}^{m_{1,i}} (n_{1,i,k} = n_{2,i,k}) \wedge (t_{1,i,k} \sim_{\mathcal{T}} t_{2,i,k})$$

**Procedures** Not sure what goes here yet.

### 2.3 $\leq_{\mathcal{T}}$

$$\alpha \leq_{\mathcal{T}} \beta \Leftrightarrow (\alpha =_{\mathcal{T}} \beta) \vee (\alpha \leq_{\mathcal{T}} \beta)$$

## 3 Primitive Types

### 3.1 Type Equality ( $=_{\mathcal{T}}$ )

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow \text{Primitive}_?( \alpha ) \wedge \text{Primitive}_?( \beta ) \wedge \alpha = \beta$$

where  $\alpha$  and  $\beta$  are types.

### 3.2 Type Equivalence ( $\sim_{\mathcal{T}}$ )

For primitive types, type equivalence is the same as type equality, so for two types  $\alpha$  and  $\beta$ :

$$\alpha \sim_{\mathcal{T}} \beta \Leftrightarrow \text{Primitive}_?( \alpha ) \wedge \text{Primitive}_?( \beta ) \wedge \alpha \leq_{\mathcal{T}} \beta$$

### 3.3 Type Assignment Compatibility ( $:=_{\mathcal{T}}$ )

$$\alpha :=_{\mathcal{T}} \beta \Leftrightarrow \text{Primitive}_?( \alpha ) \wedge \text{Primitive}_?( \beta ) \wedge \beta \leq \alpha$$

## 4 Constructed Types

### 4.1 Type Equality ( $=_{\mathcal{T}}$ )

#### 4.1.1 Arrays

$$\alpha = \text{Array}(t_1, I_1) \wedge \beta = \text{Array}(t_2, I_2)$$

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow \text{Array}_?( \alpha ) \wedge \text{Array}_?( \beta ) \wedge (t_1 =_{\mathcal{T}} t_2) \wedge ((I_1 = I_2) \vee (I_1 = \perp) \vee (I_2 = \perp))$$

#### 4.1.2 Records

$$\alpha = \text{Record}(\text{name}_1, \{(n_{1,1}, t_{1,1}), \dots, (n_{1,m_1}, t_{1,m_1})\})$$

$$\beta = \text{Record}(\text{name}_2, \{(n_{2,1}, t_{2,1}), \dots, (n_{2,m_2}, t_{2,m_2})\})$$

**Name Equality:**

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow \text{Record}_?( \alpha ) \wedge \text{Record}_?( \beta ) \wedge (\text{name}_1 = \text{name}_2)$$

## Structural Equality:

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow \text{Record}_?( \alpha ) \wedge \text{Record}_?( \beta ) \wedge (m_1 = m_2) \wedge \bigwedge_{i=1}^{m_1} (t_{1,i} =_{\mathcal{T}} t_{2,i})$$

### 4.1.3 Protocols

#### 4.1.4 Channel

The access part of a channel can be *shared*, *shared read*, *shared write*, or *not shared*.

$$\alpha = \text{Channel}(t_1, a_1) \wedge \beta = \text{Channel}(t_2, a_2)$$

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow \text{Channel}_?( \alpha ) \wedge \text{Channel}_?( \beta ) \wedge (t_1 =_{\mathcal{T}} t_2) \wedge (a_1 = a_2)$$

### 4.1.5 Channel Ends

For channel ends to be equivalent, they have to be the same ends and their channels have to be equivalent:

$$\alpha = \text{ChannelEnd}(\delta, \text{end}_1) \wedge \beta = \text{ChannelEnd}(\gamma, \text{end}_2)$$

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow \text{ChannelEnd}_?( \alpha ) \wedge \text{ChannelEnd}_?( \beta ) \wedge \text{Channel}_?( \delta ) \wedge \text{Channel}_?( \gamma ) \wedge (\text{end}_1 = \text{end}_2)$$

### 4.1.6 Procedures

$$\alpha = \text{procedure}(\text{name}_1, \{t_{1,1}, \dots, t_{1,m_1}\}, t_1) \wedge \beta = \text{procedure}(\text{name}_2, \{t_{2,1}, \dots, t_{2,m_2}\}, t_2) \wedge$$

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow (m_1 = m_2) \wedge (t_1 =_{\mathcal{T}} t_2) \wedge (\text{name}_1 = \text{name}_2) \wedge \bigwedge_{i=1}^{m_1} (t_{1,i} =_{\mathcal{T}} t_{2,i})$$

## 4.2 Type Equivalence ( $=_{\mathcal{T}} = v$ )

For all constructed types  $\alpha$  and  $\beta$ , we have:

$$\alpha \sim_{\mathcal{T}} \beta \Leftrightarrow \alpha =_{\mathcal{T}} \beta$$

## 4.3 Type Assignment Compatibility ( $:=_{\mathcal{T}}$ )

### 4.3.1 Arrays

$$\alpha = \text{Array}(t_1, I_1) \wedge \beta = \text{Array}(t_2, I_2)$$

$$\alpha :=_{\mathcal{T}} \beta \Leftrightarrow \text{Array}_?( \alpha ) \wedge \text{Array}_?( \beta ) \wedge$$

$$((\text{Protocol}_?(t_1) \wedge \text{Protocol}_?(t_2) \wedge (t_2 \leq_{\mathcal{T}} t_1)) \vee (\neg \text{Protocol}_?(t_1) \wedge \neg \text{Protocol}_?(t_2) \wedge (t_1 =_{\mathcal{T}} t_2))$$

### 4.3.2 Records

$$\alpha = \text{Record}(\text{name}_1, \{(n_{1,1}, t_{1,1}), \dots, (n_{1,m_1}, t_{1,m_1})\})$$

$$\beta = \text{Record}(\text{name}_2, \{(n_{2,1}, t_{2,1}), \dots, (n_{2,m_2}, t_{2,m_2})\})$$

$$\alpha :=_{\mathcal{T}} \beta \Leftrightarrow \alpha \sim_{\mathcal{T}} \beta$$

### 4.3.3 Protocols

### 4.3.4 Channels

Channels are non-assignable.

### 4.3.5 Channel Ends

$$\alpha = ChannelEnd(\delta, end_1) \wedge \beta = ChannelEnd(\gamma, end_2)$$

$$\begin{aligned} \alpha :=_{\mathcal{T}} \beta &\Leftrightarrow (end_1 = end_2) \wedge \\ &((Protocol_?( \delta) \wedge Protocol_?( \gamma) \wedge (\delta \leq_{\mathcal{T}} \gamma)) \vee \\ &(\neg Protocol_?( \delta) \wedge \neg Protocol_?( \gamma) \wedge (\delta =_{\mathcal{T}} \gamma)) \end{aligned}$$

### 4.3.6 Proceudres