The ProcessJ Type System

Matt Bækgaard Pedersen

October 14, 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	
short	
char	
int	
long	
float	
double	
bool	{true, false}
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	$\mathit{Array}(\alpha,\ \mathit{index})$
Record	$\mathit{Record}(\mathit{name},\{(n_1, t_1),\ldots,(n_m, t_m)\})$
Protocol	$\mathit{Protocol}(\mathit{name},\{(\mathit{tag}_1,\{(n_{1,1}, t_{1,1}),\ldots,(n_{1,m_1}, t_{1,m_1})\})$
	$\vdots$
	$(\mathit{tag}_k,\{(n_{k,1}, t_{k,1}),\ldots,(n_{k,m_k}, t_{1,m_k})\})\}$
Channel	$\mathit{Channel}(\alpha,\mathit{access})$
Channel end	$\mathit{ChannelEnd}(\alpha,\ \mathit{end})$, α is a Channel.
Procedure	$\mathit{Procedure}(\mathit{name},(t_1, t_2, \ldots, t_n),t)$

2 Helper Functions

A number of helper functions related to types are needed. These include type-versions of $<$ and $\leq$ as well as a ceiling function ($\lceil\ \rceil$). Furthermore, we need a number of *predicates*; a predicate – in this context – always has a subscript ? on it name. For example: $Numeric_?$.

2.1 $<_\mathcal{T}$

2.1.1 Primitive Types

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

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

as well as

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

but also

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

$<_\mathcal{T}$ 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

2.1.2 Constructed Types

Protocol

$$\begin{aligned}
\alpha = Protocol(name_1, \{ & (tag_{1,1}, \{(n_{1,1,1}, t_{1,1,1}), \ldots, (n_{1,1,m_{1,1}}, t_{1,1,m_{1,1}})\}), \\
& (tag_{1,2}, \{(n_{1,2,1}, t_{1,2,1}), \ldots, (n_{1,2,m_{1,2}}, t_{1,2,m_{1,2}})\}), \\
& \vdots \\
& (tag_{1,k_1}, \{(n_{1,k_1,1}, t_{1,k_1,1}), \ldots, (n_{1,k_1,m_{1,k_1}}, t_{1,k_1,m_{1,k_1}})\})\}) \\
\beta = Protocol(name_2, \{ & (tag_{2,1}, \{(n_{2,1,1}, t_{2,1,1}), \ldots, (n_{2,1,m_{2,1}}, t_{2,1,m_{2,1}})\}), \\
& (tag_{2,2}, \{(n_{2,2,1}, t_{2,2,1}), \ldots, (n_{2,2,m_{2,2}}, t_{2,2,m_{2,2}})\}), \\
& \vdots \\
& (tag_{2,k_2}, \{(n_{2,k_2,1}, t_{3,k_2,1}), \ldots, (n_{2,k_2,m_{2,k_2}}, t_{1,k_2,m_{2,k_2}})\})\})
\end{aligned}$$

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

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

Procedures Not sure what goes here yet.

2.2 $\le_{\mathcal{T}}$

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

2.3 Ceiling

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

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 α and β are types.

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

For primitive types, type equivalence is the same as type equality, so for two types α and β:

$$\alpha \sim_{\mathcal{T}} \beta \Leftrightarrow Primitive_?(\alpha) \wedge Primitive_?(\beta) \wedge \alpha \le_{\mathcal{T}} \beta$$

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

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

4 Constructed Types

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

4.1.1 Arrays

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

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

4.1.2 Records

$$\alpha = Record(name_1, \{(n_{1,1}, t_{1,1}), \ldots, (n_{1,m_1}, t_{1,m_1})\})$$

$$\beta = Record(name_2, \{(n_{2,1}, t_{2,1}), \ldots, (n_{2,m_2}, t_{2,m_2})\})$$

Name Equality:

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow Record_?(\alpha) \wedge Record_?(\beta) \wedge (name_1 = name_2)$$

Structural Equality:

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

4.1.3 Protocols

4.1.4 Channel

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

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

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

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:

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

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow ChannelEnd_?(\alpha) \wedge ChannelEnd_?(\beta) \wedge Channel_?(\delta) \wedge Channel_?(\gamma) \wedge (end_1 = end_2)$$

4.1.6 Procedures

$$\alpha = procedure(name_1, \{t_{1,1}, \ldots, t_{1,m_1}\}, t_1) \wedge \beta = procedure(name_2, \{t_{2,1}, \ldots, t_{2,m_2}\}, t_2) \wedge$$

$$\alpha =_{\mathcal{T}} \beta \Leftrightarrow (m_1 = m_2) \wedge (t_1 =_{\mathcal{T}} t_2) \wedge (name_1 = name_2) \wedge \bigwedge_{i=1}^{m_1} (t_{1,i} =_{\mathcal{T}} t_{2,i})$$

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

For all constructed types α and β, we have:

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

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

4.3.1 Arrays

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

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

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

4.3.2 Records

$$\alpha = Record(name_1, \{(n_{1,1}, t_{1,1}), \ldots, (n_{1,m_1}, t_{1,m_1})\})$$

$$\beta = Record(name_2, \{(n_{2,1}, t_{2,1}), \ldots, (n_{2,m_2}, t_{2,m_2})\})$$

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

4.3.3 Protocols

4.3.4 Channels

Channels are non-assignable.

4.3.5 Channel Ends

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

$$\begin{gathered}\alpha :=_{\mathcal{T}} \beta \Leftrightarrow (end_1 = end_2) \land \\ ((Protocol_?(\delta) \land Protocol_?(\gamma) \land (\delta \leq_{\mathcal{T}} \gamma)) \lor \\ (\neg Protocol_?(\delta) \land \neg Protocol_?(\gamma) \land (\delta =_{\mathcal{T}} \gamma))\end{gathered}$$

4.3.6 Proceudres