Register Description – *DRAFT 0.5*

Bertrand B. Blanc
bertrand.blanc@xx.com

Contents

1 Definitions

1.1 Product and Sets

A product $\mathfrak{P}$ is a couple composed of a context Γ and a root node ν: $\mathfrak{P} = (\Gamma, \nu)$. Γ is the environment of $\mathfrak{P}$, also called the context of $\mathfrak{P}$. Γ is composed of a set of forms Γ_ϕ and a set of template forms $\Gamma_{<\phi>}$ which can be instantiated through the items of Γ_ϕ: $\Gamma = \Gamma_\phi \cup \Gamma_{<\phi>} \subseteq \Phi,\ \Gamma_\phi \cap \Gamma_{<\phi>} = \varnothing$. The set Φ is the set of all the possible forms i.e. encapsulates all possible register description. The root ν is obviously an item of Γ_ϕ: $\nu \in \Gamma_\phi$.

1.2 Forms

A form $\phi \in \Gamma_\phi$ or an inner form $\phi \in \Gamma^\psi$ tuning $\psi \in \Phi$ is represented as:

$$\phi = \alpha\, (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ !A\ \underbrace{(\phi_i \mid T_i^\phi)_{i \in I_u}}_{\Gamma^\phi}$$

where

<table>
<tr><td>$\alpha \in \mathbb{A}$</td><td colspan="2">the access type. The set $\mathbb{A}$ is currently composed of 4 basic access types
1. ro read-only
2. wo write-only
3. rw read-write
4. all matches any access</td></tr>
<tr><td>$(\epsilon_i \mid T_i^\epsilon)_{i \in I_m}$</td><td colspan="2">an ordered set of extentions inherited by ϕ indexed by i</td></tr>
<tr><td></td><td>$m \in \mathbb{N}$</td><td>is the number of extensions. If $m = 0$, then the set $(\epsilon_i \mid T_i^\epsilon)_{i \in I_m}$ is reduced into $\varnothing$</td></tr>
<tr><td></td><td>$\forall i \in [\![1, m]\!],\ \epsilon_i \in \Gamma$</td><td>$\epsilon_i$ is either a form
or an instantiation of a template form</td></tr>
<tr><td></td><td>$T_i^\epsilon,\ i \in I_m^*$</td><td>is the ordered set of values given to the template form $\epsilon \in \Gamma_{<\phi>}$ to instantiate it. If $T_i^\epsilon = \varnothing$, then $\epsilon_i \in \Gamma_\phi$</td></tr>
<tr><td>$!A$</td><td colspan="2">represents the attributes of the form. The exclamation mark means that these attributes, if they are defined, are set with a final value.
Follow the current attributes $\delta,\ \rho,\ o,\ \omega$ masked by the generic A
1. $!\delta$ the description of ϕ
2. $!\rho$ the reset value of ϕ
3. $!o$ the offset of ϕ, $!o \in \mathbb{N}$
4. $!\omega$ the width of ϕ, $!\omega \in \mathbb{N}^*$</td></tr>
<tr><td>$(\phi_i \mid T_i^\phi)_{i \in I_u}$

$\phi \in \Gamma_\phi$</td><td colspan="2">an ordered set of inner forms refining the registers description of ϕ.</td></tr>
<tr><td></td><td>$u \in \mathbb{N}$</td><td>is the number of inner forms. If $u = 0$, then the set $(\phi_i \mid T_i^\phi)_{i \in I_u}$ is reduced to $\varnothing$</td></tr>
<tr><td></td><td>$\forall i \in [\![1, u]\!],\ \phi_i \in \Gamma^\phi$</td><td>$\phi_i \notin \Gamma, \phi_i \in \Gamma^\phi$.</td></tr>
<tr><td></td><td>$T_i^\phi,\ i \in I_u^*$</td><td>$T_i^\phi = \varnothing$</td></tr>
<tr><td>$(\phi_i \mid T_i^\phi)_{i \in I_u}$

$\phi \in \Gamma^\psi$</td><td colspan="2">an ordered set of inner forms refining the registers description of ψ.</td></tr>
<tr><td></td><td>$u \in \mathbb{N}$</td><td>is the number of inner forms. If $u = 0$, then the set $(\phi_i \mid T_i^\phi)_{i \in I_u}$ is reduced to $\varnothing$</td></tr>
<tr><td></td><td>$\forall i \in [\![1, u]\!],\ \phi_i \in \Gamma^\phi$</td><td>$\phi_i \notin \Gamma, \phi_i \in \Gamma^\phi$.</td></tr>
<tr><td></td><td>$T_i^\phi,\ i \in I_u^*$</td><td>is the ordered set of values given to ϕ_i to perform its inner instantiation. $T_i^\phi = \varnothing$ if $\nexists \psi \in \Gamma_{<\phi>} \,/\, \psi \rightarrow^* \phi$
$T_i^\phi = \varnothing$ if $\phi \in \Gamma_\phi$</td></tr>
</table>

Property 1: contiguity coherency

Be $\psi \ = \ \alpha \varnothing \ A \ (\phi_i)_{i \in I_u^*}, \ \psi \in \Phi$ with $o \in A$ and $\omega \in A$.

Then, once the width attribute is set within a form, it must be equal to the sum of the widths recursively computed on the inner-forms tuning this form.

$$\omega \ = \ \sum_{i=1}^{u \in \mathbb{N}^*} \phi_i.\omega$$

However, we sometimes need to let blank bit-fields to capture for example remote registers. These special bit-fields cannot indeed be tuned with inner forms.

Property 2: range coherency

Two inner-forms refining a form cannot overlap.

Be $\psi \ = \ \alpha \varnothing \ A \ (\phi_i)_{i \in I_u^*}, \ \psi \in \Phi$.

We need to introduce here a bounded range $[\lambda_b, \ \lambda_e]$ in which ψ evolves i.e. ψ set bits, not necessary contiguous within the range $[\lambda_b, \ \lambda_e]$. Such a range is noted $[\psi.\lambda_b, \ \psi.\lambda_e]$.

Then, $\forall i \in I_u^*, \ \forall j \in I_u^* \backslash \{i\}, \ [\phi_i.\lambda_b, \ \phi_i.\lambda_e] \ \cap \ [\phi_j.\lambda_b, \ \phi_j.\lambda_e] \ = \ \varnothing$

Property 3: overlap coherency

The above property 2 avoids any overlap between two forms. However, the form overlap is sometimes needed. Two behaviors have got exhibited.

Property 3.1: dual forms

Two forms $\phi_1 \in \Phi$ and $\phi_2 \in \Phi \backslash \{\phi_1\}$ are dual if and only if

(i)	the both inner-forms are not refined	$\Gamma^{\phi_1} \ = \ \Gamma^{\phi_2} \ = \ \varnothing$
(ii)	the both inner-forms have the same defined offset	$\phi_1.o \ = \ \phi_2.o$
(iii)	the both inner-forms have the same defined width	$\phi_1.\omega \ = \ \phi_2.\omega$
(iv)	the both inner-forms refine the same form	$\exists \psi \in \Phi \ / \ \phi_1 \in \Gamma^{\psi} \ \wedge \ \phi_2 \in \Gamma^{\psi}$

Property 3.2: modal forms

We need to introduce here a modal attribute $\beta \in \mathbb{M}$ in order to tag a form $\phi \in \Phi$ as follow:

$$\phi \ = \ \beta \, \alpha \varnothing \ !A \ (\phi_i)_{i \in I_u}$$

Two forms $\phi_1 \in \Phi$ and $\phi_2 \in \Phi \backslash \{\phi_1\}$ are modal if and only if

(i)	modes are identical	$\phi_1.\beta \ \neq \ \phi_2.\beta$
(ii)	offsets are the same	$\phi_1.o \ = \ \phi_2.o$
(iii)	widths are the same	$\phi_1.\omega \ = \ \phi_2.\omega$
(iv)	the both modal inner-forms refine the same form	$\exists \psi \in \Phi \ / \ \phi_1 \in \Gamma^{\psi} \ \wedge \ \phi_2 \in \Gamma^{\psi}$

1.3 Template Forms

A template form $< \phi > \in \Gamma_{<\phi>}$ is represented as:

$$< \phi > = \ (\tau_i)_{i \in I_n} \ \alpha \ (\epsilon_i \mid T_i^{\epsilon})_{i \in I_m} \ A \ \underbrace{(\phi_i \mid T_i^{\phi})_{i \in I_u}}_{\Gamma^{\phi}}$$

where

$(\tau_i)_{i \in I_n}$	the ordered set of final abstract items (basic τ_i) or constrained abstract item (τ_i matches $\chi \ \rightarrow \ \tau_i, \ \chi \in \Gamma$ means that when instantiate, this item must be compliant with χ) needed in the body of $< \phi >$ to instantiate it. $n \neq 0$ since $< \phi > \in \Gamma_{<\phi>}$.
$\alpha \in \mathbb{A}$	the access type. The set $\mathbb{A}$ is currently composed of 4 basic access types 1. ro read-only 2. wo write-only 3. rw read-write 4. all matches any access
$(\epsilon_i \mid T_i^{\epsilon})_{i \in I_m}$	an ordered set of extentions inherited by $< \phi >$ indexed by i
$m \in \mathbb{N}$	is the number of extensions. If $m = 0$, then the set $(\epsilon_i \mid T_i^{\epsilon})_{i \in I_m}$ is reduced into $\varnothing$
$\forall i \in [\![1, m]\!], \ \epsilon_i \in \Gamma$	ϵ_i is either a form or an instantiation of a template form or an abstract item $\tau \in (\tau_i)_{i \in I_n}$ aimed to be instantiated with an item of Γ.
$T_i^{\epsilon}, \ i \in I_m^*$	is the ordered values or abstract values $\tau \in (\tau_i)_{i \in I_n}$ given to the template form $\epsilon_i \in \Gamma_{<\phi>}$ to instantiate it. If $T_i^{\epsilon} = \varnothing$, then either $\epsilon_i \in \Gamma_{\phi}$ or $\epsilon_i \in (\tau_i)_{i \in I_n}$. If $\epsilon_i \in (\tau_i)_{i \in I_n}$ and $\tau_i \ \chi - constrained$, then ϵ_i is $\chi - typed$
A	represents the attributes of the template form. These attributes, if they are defined, are set with either a final value or an abstract value $\tau \in (\tau_i)_{i \in I_n}$. Follow the current attributes $\delta, \ \rho, \ o, \ \omega$ masked by the generic A 1. $[\delta \mid \tau]$ the abstract description of $< \phi >, \tau \in (\tau_i)_{i \in I_n}$ 2. $[\rho \mid \tau]$ the abstract reset value of $< \phi >, \tau \in (\tau_i)_{i \in I_n}$ 3. $[o \mid \tau]$ the offset of $< \phi >, \tau \in (\tau_i)_{i \in I_n}$ 4. $[\omega \mid \tau]$ the width of $< \phi >, \tau \in (\tau_i)_{i \in I_n}$ 5. $!\delta$ the instantiated description of $< \phi >$ 6. $!\rho$ the instantiated reset value of $< \phi >$ 7. $!o$ the instantiated offset of $< \phi >, o \ \in \ \mathbb{N}$ 8. $!\omega$ the instantiated width of $< \phi >, \omega \ \in \ \mathbb{N}^*$
$(\phi_i \mid T_i^{\phi})_{i \in I_u}$	an ordered set of inner forms refines the registers description of $< \phi >$.
$u \in \mathbb{N}$	is the number of inner forms. If $u = 0$, then the set $(\phi_i \mid T_i^{\phi})_{i \in I_u}$ is reduced to $\varnothing$
$\forall i \in [\![1, u]\!], \ \phi_i \in \Gamma^{\phi}$	$\phi_i \in \Gamma^{\phi}$, where Γ^{ϕ} represents the set of inner refinements.
$T_i^{\phi}, \ i \in I_u^*$	is the oredered set composed of either final values or abstract value $\tau \in (\tau_i)_{i \in I_n}$ given to ϕ_i to perform the instantiation of inner components recursively. $T_i^{\phi} = \varnothing$ if $(\tau_i)_{i \in I_n}$ do not impact Γ^{ϕ}

1.3.1 Inheritance Status

Some inner forms $\phi_i \in \Gamma^\phi$ are ought to mandatory be defined when inherited. To ensure that the form will be refined in the caller, the form is declared in the callee as an abstract inner form. Such a form is noted in the callee as $\hat{\phi}_i \in \Gamma^\phi$.

1.4 Miscellaneous

1.4.1 Sets

1. $C \ = \ A \cup B \ \equiv \ (\forall x \in C, \ x \in A \vee x \in B) \ \wedge \ (\forall x \in A, \ \forall y \in B, \ \{x, \ \} \in C^2)$
2. $C \ = \ A \cap B \ \equiv \ (\forall x \in C, \ x \in A \wedge x \in B) \ \wedge \ (\forall x \in A \ / \ x \in B, \ x \in C)$
3. $A \cap B \ = \ \varnothing \ \equiv \ \forall x \in A, \ \forall y \in B, \ x \notin B \wedge y \notin A$
4. $(x_i)_{i \in I_n}$
   (i) $i \ = \ 0 \ \Rightarrow \ (x_i)_{i \in I_n} \ = \ \varnothing$
   (ii) $i \ \neq \ 0 \ \Rightarrow \ (x_i)_{i \in I_n} \ = \ (x_i)_{i \in I_n^*}$
   (iii) is an ordered set (if defined) indexed by $i \in [\![1, \ n]\!]$: $\{ \ x_1, \ x_2, \ldots, \ x_n \ \}$

1.4.2 Reductions & Expansions

1. The form a is reducted into b in a single step, applying a rule. Seven rules are defined beyond in the section devoted to the Model of Computation.
$$a \ \rightarrow \ b \ \equiv \ \exists t \in \{x, \ r, \ d, \ \epsilon, \ \mu, \ \tau, \ \sigma\} \ / \ a \ \rightarrow_t \ b$$
2. The form a is reducted into b in at least one step.
$$a \ \rightarrow^+ \ b \ \equiv \ \exists c \in \Phi \ / \ a \ \rightarrow \ c \ \rightarrow^* \ b$$
3. The form a is either reducted into b in at least one step, or is already reducted.
$$a \ \rightarrow^* \ b \ \equiv \ \exists n \in \mathbb{N}, \ \exists (c_i)_{i \in \mathbb{N}} \ /$$
   (i) $n \ = \ 0 \ \Rightarrow \ a \ \sim \ b$
   (ii) $n \ \neq \ 0 \ \Rightarrow \ (\ a \ \rightarrow^+ \ b \equiv a \ \rightarrow \ c_1 \ \rightarrow \ \ldots \ \rightarrow \ c_n \ \rightarrow \ b)$
4. The form a is reducted into its fully reducted normal form in either at least one step, or is already reducted.
$$a \ \rightarrow^* \ !a \ \equiv \ (\nexists \phi \in \Phi \ / \ !a \ \rightarrow \ \phi) \ \wedge \ (!a \ \not\sim \phi)$$
5. The couple of forms a and b are equivalent each other modulus basic expansion and reduction rules.
$$a \ \sim \ b \ \equiv \ a \ \rightarrow_{x,r} \ b$$
6. Corollary of the item above.
$$a \ \not\sim b \ \equiv \ \exists t \in \{ \ d, \ \epsilon, \ \mu, \ \tau, \ \sigma\} \ / \ a \ \rightarrow_t \ b$$

1.4.3 Substitutions

The substitution operator is basically noticed $[\ X \mid Y \ ]$ where x is a set to get substituted and Y a set of items of substitution.

1. the semantics of this operator depends on the type of the sets represented by X and Y
2. a singleton can either be noticed $\{x\}$ or x if a set is expected according to the type-check. Discarding braces prevents verbosity
3. if sets size is greater than 1, a binary associative law must be defined

1.5 Example

Follow five examples to highlight the definitions above:

1. $\mathfrak{P} = (\Gamma, \phi_1),\ \phi_1 \in \Gamma_\phi$
   The product $\mathfrak{P}$ is composed of a set Γ of forms (Γ_ϕ) and a set of template forms ($\Gamma_{<\phi>}$). Γ is the union of Γ_ϕ and $\Gamma_{<\phi>}$. The node ϕ_1 is the root and is obviously an item of Γ_ϕ.

2. $\Gamma_\phi = \{\phi_1\},\ \Gamma_{<\phi>} = \{\phi_2\},\ \Gamma = \Gamma_\phi \cup \Gamma_{<\phi>} = \{\phi_1,\ \phi_2\},\ \Gamma_\phi \cap \Gamma_{<\phi>} = \varnothing$
   The set of forms Γ_ϕ is a singleton only composed of ϕ_1. The set of template forms $\Gamma_{<\phi>}$ is a singleton only composed of ϕ_2. Γ is therefore the set composed of these both items ϕ_1 and ϕ_2.

3. $\phi_2 = \{\tau\}\ \alpha_2\ \varnothing\ [\rho|\tau]\ \varnothing,\ \alpha_2 \in \mathbb{A}$

   ϕ_2 is a template form and thus matches the representation

   $$(\tau_i)_{i\in I_n}\ \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i\in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i\in I_u}$$

   - $(\tau_i)_{i\in I_n} = \{\tau\}$, then $n = 1$, and $\tau_1 = \tau$
   - $\alpha = \alpha_2$ is obviously a valid access type since $\alpha_2 \in \mathbb{A}$
   - $(\epsilon_i \mid T_i^\epsilon)_{i\in I_m} = \varnothing$, then $m = 0$
   - $A = [\rho|\tau]$, then the reset value ρ is abstracted and will be instantiated as soon as the template form $< \phi >$ will be instantiated with obviously one parameter τ. This template form has no description δ, no offset o and no width ω
   - $(\phi_i \mid T_i^\phi)_{i\in I_u} = \varnothing$, means $u = 0$ and $\Gamma^{\phi_2} = \varnothing$.

4. $\phi_1 = \alpha_1\ \{\phi_2|\kappa\}\ !\delta_1\ \{\phi_{11}\},\ \alpha_1 \in \mathbb{A},\ \phi_{11} \in \Gamma^{\phi_1},\ \kappa$ is a reset value

   $$\alpha\ (\epsilon_i \mid T_i^\epsilon)_{i\in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i\in I_u}$$

   - $\alpha = \alpha_1$ is obviously a valid access type since $\alpha_1 \in \mathbb{A}$
   - $(\epsilon_i \mid T_i^\epsilon)_{i\in I_m} = \{\phi_2|\kappa\}$, then $m = 1$, $\epsilon_1 = \phi_2 \in \Gamma$ and $T_1^\epsilon = \kappa$ which is a reset value. $T_1^\epsilon \neq \varnothing$ ensures that $\epsilon_1 = \phi_2 \in \Gamma_{<\phi>} \subseteq \Gamma$.
     Notice that if $T_1^\epsilon = \varnothing$, then the context would have been violated.
     - First demonstration

       (i) by assumption, $\phi_2 \notin \Gamma_\phi$
       (ii) $\phi_2 \notin \Gamma_{<\phi>}$ since $T_1^\epsilon = \varnothing$
       (iii)then $\phi_2 \notin \Gamma_\phi \cup \Gamma_{<\phi>}$
       (iv)then $\phi_2 \notin \Gamma \# \forall \epsilon \in (\epsilon_i)_{i\in I_m},\ \epsilon \in \Gamma$
     - Second demonstration

       (i) by assumption, $\phi_2 \in \Gamma_{<\phi>}$
       (ii) $\phi_2 \notin \Gamma_{<\phi>}$ since $T_1^\epsilon = \varnothing$
       (iii)then a contradiction is reached $\#$

     Moreover, ϕ_2 is a template form which will be instantiated with one item. Indeed, ϕ_2 has a single abstract description field, which is coherent typed with κ.
   - $A = !\delta_1$ ensures that ϕ_1 has only a final description field, with neither any offset o, nor reset value ρ, nor width ω. The sensitive reader would have right noticed the a reset value would have been indeed inherited from ϕ_2. However, The Model of Computation is not focused within this section.

- $(\phi_i \mid T_i^{\phi})_{i \in I_u} = \{\phi_{11}\}$, means that $u = 1$, $\phi_1 = \phi_{11}$ (take care, this equation means that ϕ_{11} matches ϕ_1 which is here the first item of the inner forms set $(\phi_i \mid T_i^{\phi})_{i \in I_u} = \Gamma^{\phi_1}$ of the form $\phi_1 \in \Gamma_{\phi}$) and $T_1^{\phi} = \varnothing$. The set Γ^{ϕ_1} of inner forms is declared as $\Gamma^{\phi_1} = \{\phi_{11}\}$ to tune the description of the form ϕ_1. $T_1^{\phi} = \varnothing$ ensures that this inner form do not need to instantiate deeper abstract items coming from ϕ_1.

5. $\phi_{11} = \alpha_{11} \varnothing\, !\delta_{11}\, !o_{11}\, \varnothing,\ \alpha_{11} \in \mathbb{A}$
   $\phi_{11} \in \Gamma^{\phi_1}$ is a valid inner form, matching the same semantics depicted above for ϕ_1

2 Model of Computation

For a product $\mathfrak{P} = (\Gamma, \nu),\ \nu \in \Gamma_\phi$, this **M**odel **o**f **C**omputation aims to reduce ν into a reducted normal form $!\nu$ fully instantiated, following a set of reduction and expansion rules defined below. This MoC applied on ν is noted:

$$< \nu >^+ \rightarrow^* !\nu,\ \nu \in \Gamma_\phi$$
$$< \nu | T >^+ \rightarrow^* !\nu,\ \nu \in \Gamma_{<\phi>},\ T \text{ an ordered set of parameters needed to instantiate } \nu$$

2.1 Rule 1: x-expansion $\rightarrow_x$

The expansion is a syntactical rewriting wich expands an abstract item into an expression in order to perform further operation.

2.1.1 Expansion of a form $\phi \in \Gamma_\phi$

$$\phi = \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u},\ \phi \in \Gamma_\phi$$
$$< \phi >^+ \rightarrow_x < \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} >^+$$

also noticed:

$$\frac{< \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} >^+}{< \phi >^+}$$

2.1.2 Expansion of a template form $< \phi > \in \Gamma_{<\phi>}$

$$< \phi > = (\tau_i)_{i \in I_n}\ \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u},\ \phi \in \Gamma_{<\phi>}$$
$$< \phi \mid T >^+ \rightarrow_x < (\tau_i)_{i \in I_n}\ \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} \mid T >^+$$

also noticed:

$$\frac{< (\tau_i)_{i \in I_n}\ \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} \mid T >^+}{< \phi \mid T >^+}$$

2.2 Rule 2: d-expansion (distributivity) $\rightarrow_d$

The distributivity is a syntactical rewriting wich will inject an operation inside an expression in order to perform this operation recursively.

2.2.1 Distributivity on an expanded form $\phi \in \Gamma_\phi$

$$< \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} >^+ \rightarrow_d < \alpha\ (< \epsilon_i \mid T_i^\epsilon >^+)_{i \in I_m}\ A\ (< \phi_i \mid T_i^\phi >^+)_{i \in I_u} >^+$$

also noticed:

$$\frac{< \alpha\ (< \epsilon_i \mid T_i^\epsilon >^+)_{i \in I_m}\ A\ (< \phi_i \mid T_i^\phi >^+)_{i \in I_u} >^+}{< \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} >^+}$$

2.2.2 Distributivity on an expanded template form $< \phi >\in \Gamma_{<\phi>}$

This rule is unforeseen and illegal. Indeed, distributing the operator $< . >^+$ inside the template form is unrelevant since some T_i^ϵ or T_i^ϕ can be abstracted. This operation could only be perform after getting all abstract item instantiated with T. See the instantiation rule below. The template form will be computed into an instantiated form and we will therefore be able to apply the same distributivity rule as the one defined above for $\phi \in \Gamma_\phi$

2.3 Rule 3: r-reduction $\rightarrow_r$

The reduction is a syntactical rewriting wich reduces a computed expression into an abstract item in order to meet a known syntactical expression.

(i) inheritance reduction $< \epsilon_i | T_i^\epsilon >^+ \rightarrow_r !\epsilon_i$ also noticed $\dfrac{!\epsilon_i}{< \epsilon_i | T_i^\epsilon >^+}$

Notice that here is matched one and only one rule of the couple of basic rules $< \nu >^+ \rightarrow !\nu,\ \nu \in \Gamma_\phi$ if $T_i^\epsilon = \varnothing$, or $< \nu | T >^+ \rightarrow !\nu,\ \nu \in \Gamma_{<\phi>}$.

(ii) inner form reduction $< \phi_i | T_i^\phi >^+ \rightarrow_r !\phi_i$

Notice that here is matched the basic rule $< \nu | T >^+ \rightarrow !\nu,\ \nu \in \Gamma^\phi$.

(iii) the inheritance and instantiation rules will be deeper explain in their respective subsection.

type	expansion	reduction
inheritance	ϵ-expansion	μ-reduction
instantiation	τ-expansion	σ-reduction

2.4 Rule 4: inheritance

2.4.1 ϵ-expansion $\rightarrow_\epsilon$

This ϵ-expansion is only managed under the following assumptions:

(i) the inherited forms are reduced into a reducted normal form $!\epsilon_i,\ \epsilon_i \in \Gamma$

(ii) the attributes are fully instantiated $!A$

(iii) the inner forms are reduced into a reducted normal form $!\phi_i,\ \phi_i \in \Gamma^\phi$

(iv) ϕ is either a form, or an inner-form, or an instantiated form.

$$\phi \in \{\phi' \ / \ (\phi' \in \Gamma_\phi) \ \vee \ (< \psi | T >^+ = \ \phi',\ \psi \in \Gamma_{<\phi>}) \ \vee \ (\phi' \in \Gamma^\phi)\}$$

$$< \ \alpha \ (!\epsilon_i)_{i \in I_m} \ !A \ (!\phi_i)_{i \in I_u} >^+ \rightarrow_\epsilon \ [\alpha \mid (!\epsilon_i)_{i \in I_m}] \varnothing [!A \mid (!\epsilon_i)_{i \in I_m}] \ [(!\phi_i)_{i \in I_u} \mid (!\epsilon_i)_{i \in I_m}]$$

$$\frac{[\alpha \mid (!\epsilon_i)_{i \in I_m}] \varnothing [!A \mid (!\epsilon_i)_{i \in I_m}] \ [(!\phi_i)_{i \in I_u} \mid (!\epsilon_i)_{i \in I_m}]}{< \ \alpha \ (!\epsilon_i)_{i \in I_m} \ !A \ (!\phi_i)_{i \in I_u} >^+}$$

where

$[\alpha \mid (!\epsilon_i)_{i \in I_m}]$**:** the reduced access type $\alpha' \in \mathbb{A}$ is associatively computed according to the primal access type α of ϕ and the inherited ones as: $[\alpha \mid (!\epsilon_i)_{i \in I_m}] \ = \ [\ldots [[\alpha \mid !\epsilon_1] | !\epsilon_2] \ldots | !\epsilon_m]$

α	$!\epsilon_i.\alpha,\ i \in I_m^*$	$\alpha' = [\alpha \mid !\epsilon_i.\alpha]$
all	all	all
all	ro	ro
all	wo	wo
all	rw	rw
ro	all	ro
ro	ro	ro
ro	wo	$\bot$
ro	rw	$\bot$
wo	all	wo
wo	ro	$\bot$
wo	wo	wo
wo	rw	$\bot$
rw	all	rw
rw	ro	$\bot$
rw	wo	$\bot$
rw	rw	rw

$\varnothing$**:** Inherited forms did obviously disapper since they have been merged within ϕ.

$[!A \mid (!\epsilon_i)_{i\in I_m}]$**:** The attributes are associatively computed, merging the attributes of ϕ with the ones inherited as depicted below. $[!A \mid (!\epsilon_i)_{i\in I_m}] = [\ldots[\,[!A|!\epsilon_1]\,|\epsilon_2]\ldots]$.

$!A$	$!\epsilon_i.!A,\ i \in I_m^*$	$!\epsilon_j.!A,\ j \in I_m^* \setminus \{i\}$	$!A'$	comment
$\top$	$\{\top,\ \bot\}$	$\{\top,\ \bot\}$	$!A$	overload
$\bot$	$\top$	$\top$	$\bot$	inheritance conflict
$\bot$	$\top$	$\bot$	$!\epsilon_i.!A$	inheritance
$\bot$	$\bot$	$\bot$	$\bot$	undefined

$[(!\phi_i)_{i\in I_u} \mid (!\epsilon_i)_{i\in I_m}]$**:** The set of inner forms will be grown up with the inherited inner forms coming from $(\epsilon_i)_i$ as depicted below.

$!\phi_i \in \Gamma^\phi$ $i \in I_u^*$	$!\epsilon_i.!\phi \in \Gamma^{!\epsilon_i}$ $i \in I_m^*$	$!\epsilon_j.!\phi \in \Gamma^{!\epsilon_j}$ $j \in I_m^* \setminus \{i\}$	$!\phi'$	comment
$\bot$	$\top$	$\bot$	$!\epsilon_i.!\phi$	inheritance $!\epsilon_i.!\phi \in \Gamma^\phi$
$\bot$	$\bot$	$\bot$	$\bot$	undefined
$\bot$	$\top$	$\top$	$\bot$	conflict
$\top$	$\top$	$\bot$	$\mu[!\phi_i\|!\epsilon_i.!\phi]$	inheritance merge (μ-reduction)
$\top$	$\bot$	$\bot$	$!\phi_i$	basic definition
$\top$	$\top$	$\top$	$\mu[!\phi_i\|!\epsilon_i.!\phi,\ !\epsilon_j.!\phi]$	inheritance merge (μ-reduction)

Mandatory Inner Forms $\hat{\psi} \in \Gamma^\phi$**:** The mandatory inner forms are expected to be defined when inherited, following the rule:

$!\psi \in \Gamma^\phi$	$!\epsilon.!\hat{\phi} \in \Gamma^{!\epsilon}$	status	comment
$\top$	$\top$	$\top$	declared in the callee and defined in the caller
$\top$	$\bot$	$\top$	no mandatory inner form declared in the callee
$\bot$	$\top$	$\bot$	declared in the callee but nor defined in the caller
$\bot$	$\bot$	$\top$	neither declared nor defined

2.4.2 μ-reduction $\rightarrow_\mu$

$\mu[!\phi_a|!\phi_b]$ and $\mu[!\phi_a|!\phi_b,\ !\phi_c]$ are the merging operator defined as follow, computing a merged inner instance form $!\phi_\mu \in \Gamma^\phi$, either overloaded, or refined.

$\mu[!\phi_a\|!\phi_b,\ !\phi_c]$	(i)	$!\phi_\mu.\alpha$	$=$	$[!\phi_a.\alpha\|!\phi_b.\alpha,\ !\phi_c.\alpha]$	access type definition and refinement
	(ii)	$!\phi_\mu.!A$	$=$	$[!\phi_a.!A\|\phi_b.!A,\ !\phi_c.!A]$	attributes definition
	(iii)	$!\phi_\mu.\Gamma^{\phi_\mu}$	$=$	$[!\phi_a.\Gamma^{\phi_a}\|!\phi_b.\Gamma^{\phi_b},\ \phi_c.\Gamma^{\phi_c}]$	inner forms definition
$\mu[!\phi_a\|!\phi_b]$	(iv)	$!\phi_\mu.\alpha$	$=$	$[!\phi_a.\alpha\|!\phi_b.\alpha]$	access type definition and refinement
	(v)	$!\phi_\mu.!A$	$=$	$[!\phi_a.!A\|!\phi_b.!A]$	attributes definition
	(vi)	$!\phi_\mu.\Gamma^{\phi_\mu}$	$=$	$[!\phi_a.\Gamma^{\phi_a}\|!\phi_b.\Gamma^{\phi_b}]$	inner forms definition

2.4.3 Access refi nement in inner forms

The access mode of the inner forms $\psi \in \Gamma^\phi$ is also refined in order to be aligned with the access type of ϕ. The following table define the access refinement.

$\phi.\alpha$	$\psi.\alpha$	$[\phi.\alpha\|\psi.\alpha]$
all	all	all
all	ro	ro
all	wo	wo
all	rw	rw
ro	all	ro
ro	ro	ro
ro	wo	$\perp$
ro	rw	ro
wo	all	wo
wo	ro	$\perp$
wo	wo	wo
wo	rw	wo
rw	all	rw
rw	ro	ro
rw	wo	wo
rw	rw	rw

2.4.4 r-reduction

Consequently, we can thereafter perform a syntactical r-reduction:

$$[\alpha \mid (!\epsilon_i)_{i\in I_m}] \oslash [!A \mid (!\epsilon_i)_{i\in I_m}]\ [(!\phi_i)_{i\in I_u} \mid (!\epsilon_i)_{i\in I_m}] \rightarrow_r \alpha' \oslash !A'\ (!\phi'_i)_{i\in I_{u'}}$$

$$\frac{\alpha' \oslash !A'\ (!\phi'_i)_{i\in I_{u'}}}{[\alpha \mid (!\epsilon_i)_{i\in I_m}] \oslash [!A \mid (!\epsilon_i)_{i\in I_m}]\ [(!\phi_i)_{i\in I_u} \mid (!\epsilon_i)_{i\in I_m}]}$$

2.5 Rule 5: instantiation

The τ-expansion rule aims to inject values within a template form $\phi \in \Gamma_{<\phi>}$ in order to instantiate it.

$$< (\tau_i)_{i\in I_n}\ \alpha\ (\epsilon_i \mid T_i^{\epsilon})_{i\in I_m}\ A\ (\phi_i \mid T_i^{\phi})_{i\in I_u} \mid T >^{+} \rightarrow_{\tau} < \alpha\ (\epsilon_i \mid \sigma(T_i^{\epsilon}, T))_{i\in I_m}\ \sigma(A, T)\ (\phi_i \mid \sigma(T_i^{\phi}, T))_{i\in I_u} >^{+}$$

$$\frac{< \alpha\ (\epsilon_i \mid \sigma(T_i^{\epsilon}, T))_{i\in I_m}\ \sigma(A, T)\ (\phi_i \mid \sigma(T_i^{\phi}, T))_{i\in I_u} >^{+}}{< (\tau_i)_{i\in I_n}\ \alpha\ (\epsilon_i \mid T_i^{\epsilon})_{i\in I_m}\ A\ (\phi_i \mid T_i^{\phi})_{i\in I_u} \mid T >^{+}}$$

where

$(\epsilon_i \mid (\sigma(T_i^{\epsilon}, T))_{i\in I_m}$**:** The inherited forms instantiation setting up the abstract values within the templates forms.

example	$T_i^{\epsilon},\ i \in I_m$	$\sigma(T_i^{\epsilon}, \sigma'(T))$
1	$\varnothing \Leftrightarrow m = 0$	$\varnothing$
2	κ a final instantiated value	κ
3	$T' \in \Gamma_{\phi}$	$< T' >^{+}$
4	$T' \in \Gamma_{<\phi>}$	$< T' \mid T'' \subseteq \sigma'(T) >^{+},\ \#T'' = \#T'.(\tau_i)_i$
5	$\tau \in (\tau_i)_{i\in I_n}$	$!\theta\ /\ \exists j \in I_n^*,\ \exists k \in I_{\#T_i^{\epsilon}},\ \exists\ t'_k \in T_i^{\epsilon} = \tau_j \in (\tau_i)_i$ $\wedge\ T = (t_i'')_{I_{\#T}^*},\ t_j'' = \theta$ $\wedge\ \theta \rightarrow^* !\tau$
6	$(\psi \rightarrow \tau) \in (\tau_i)_{i\in I_n}$	$!\theta\ /\ \exists j \in I_n^*,\ \exists k \in I_{\#T_i^{\epsilon}},\ \exists\ t'_k \in T_i^{\epsilon} = \tau_j \in (\tau_i)_i$ $\wedge\ T = (t_i'')_{I_{\#T}^*},\ t_j'' = \theta$ $\wedge\ !\theta \sim \psi$

demonstration	T	$\sigma'(T)$
	$\varnothing$	$\varnothing$
	κ a final instantiated value	κ
	$(t_i)_i$	$(\sigma'(t_i))_i$
	$T' \in \Gamma_{\phi}$	T' already reduced
1	$T' \in \Gamma_{<\phi>}$	T' already reduced. This case is hence illegal.

2.5.1 Examples

This section practicaly highlights the mathematical model above. The following set Γ composed of template-forms, instance-forms and inner-forms will be used in each examples.

```
RegW  = all {} (w=32) {}
range = {MAX, MIN, RESET} all {RESET} [o|MIN] [w|MAX-MIN+1] {}
dflR  = all {} (r=0x0) {}
reset = {RESET} all {} [r=RESET] {}
REV   = {MAX, MIN} ro {RegW | {}} { pad | {}, Max | {MAX}, Min | {MIN} }
pad   = ro {range | {32, 8, dflR}} {}
Max   = ro {range | {7, 4, reset | {MAX}}} {}
Max   = ro {range | {3, 0, reset | {MIN}}} {}
```

Example 1 The inherited form is an instance-form.

In REV template-form, consider the piece $\{RegW \mid \varnothing\}$.

$REV.I_m = \varnothing$, then $m = 0$ and obviously $T^{\epsilon} = \varnothing$.

Example 2 The inherited form is a template form which needs a single parameter to perform its instantiation. This parameter is moreover a final value (i.e. is not an element of Φ).

In pad instance form, consider $\{range \mid \{32, \ldots\}\}$.

$m = 2, T^{\epsilon} = \{32, \ldots\}$, then T_1^{ϵ} is obviously a final instantiated value.

Example 3 The inherited form is a template form taking a single parameter which is an instance-form.

In pad instance form, consider $\{range \mid \{\ldots,\ dflR\}\}$.

$dflR \in \Gamma_\phi$ is the 3^{th} parameter of instantiation of the template-form $range$. It will be self reduced before be instantiated within $range$.

Example 4 The inherited form is a template form taking a single template-form parameter. This template-form parameter obviously needs to be itself instantiated with parameters. These parameters are either hard set (examples 2 and 3) or come from the parameters given to the primary template-form.

In Max instance form, consider $\{range \mid \ \ldots,\ reset \mid \{MAX\}\}\}$.

$reset \in \Gamma_{<\phi>}$ must be instantiated before be inherited by Max. The parameters it needs to perform its instantiation are a subset of the given parameters coming from the refined form. Max refines REV which is a template-form taking MAX as abstract parameter. MAX is needed within Max.

To instantiate $range$, consider the notation $< range \mid \{7,\ 4,\ reset \mid \{MAX\}\} >$, and hence, the 3^{th} parameter $RESET$ is replaced with $reset \mid \{MAX\},\ reset \in \Gamma_{<\phi>}$.

Besides, the abstract parameter of $reset$ will be instantiated with MAX.

$reset$ is instantiated with the single parameter it needs: MAX. The needed parameters are a subset of the parameters given to the form to perform its instantiation. In this case, the needed parameter MAX is the only one given to Max thru REV. MAX is a final instantiated value (second raw in the second table).

Example 5 The inherited form is an abstract parameter which will be dynamically inherited.

In REV instance form, taking two parameters χ_1 and χ_2 (in this order) to be instantiated, consider $Min \mid MIN$.

$REV \in \Gamma_{<\phi>}$, then before performing the instantiation of Min, the given parameter χ_2 is first of all full reducted without no significant endeavor since it must be a final value. But why is χ_2 chosen?

Be $j \in I_2^*,\ j = 2$, be $k \in I^*_{\sharp T_i^\epsilon} = T^\epsilon_{\sharp\{MIN\}} = T_1^\epsilon,\ k = 1$.

Consequently, $\exists t_2 \in (\tau_i),\ t_2 = MIN, \exists t_1' \in T_1^\epsilon,\ t_1 = MIN$ as $t_2 = t_1'$.

Moreover, the j^{th} item of T is χ_2, since $j = 2$.

To conclude, Min is instantiated with $!\chi_2$, once χ_2 becomes itself full instantiated.

Example 6 The inherited form is an abstract parameter which will be dynamically inherited (example 5). Moreover, this abstract parameter is constrained with a form in Γ, to ensure, at run-time, that the given parameter is compliant with the constrained-form.

Be $range = \{MAX,\ MIN,\ RESET\}\ all\ \{RESET\}[o \mid MIN][w \mid MAX - MIN + 1]\varnothing$ modified into $range = \{MAX,\ MIN,\ reset \rightarrow RESET\}\ all\ \{RESET\}[o \mid MIN][w \mid MAX - MIN + 1]\varnothing$.

The way of computation is the same as explained above in the 5^{th} example in order to instantiate the template-form $range$ where the 3^{th} abstract parameter needs to be instantiate first.

The $RESET$ given parameter when an instantiation is performed needs to be itself an instantiation of the template-form $reset$.

2.5.2 Demonstrations

Demonstration 1 Demonstrate that a template-form given as parameter to another template-form is already full reduced.

(i) be $(\Gamma,\ \nu),\ \nu \in \Gamma_\phi$ a product

(ii) be $< \phi_x | T >^+ = < \{\tau\}\ \alpha\ \{\epsilon \mid \tau\} \ldots | \{\tau'\} >^+,\ \tau' \in \Gamma_{<\phi>}$ and τ' is not reduced

(iii) $\exists s \in \Gamma_\phi\ /\ \underbrace{\nu}_{\in \Gamma_\phi} \rightarrow \ldots \rightarrow \underbrace{s}_{\in \Gamma_\phi} \rightarrow \ldots \rightarrow \underbrace{\phi_x}_{\in \Gamma_{<\phi>}}$

(iv) Then, $\exists\ (\phi_{x_i})_{i \in I_n} \in \Gamma^n_{<\phi>}\ /\ \phi_{x_n} \mid \{\phi_{x_{n-1}} \mid \{\ldots | \{\phi_{x_1} \mid \{\phi_{x_0} \mid \{\lambda \mid \varnothing\}\}\} \ldots\}\}$, with $\phi_{x_n} = \phi_x$ and $s = \alpha\ \{\ \phi_{x_0} \mid \{\lambda \mid \varnothing\ \}\}\ A\ (\phi_i)_i$

(v) The σ-reduction is derivated in 6 separate cases. We assume that the assumption is wrong.

(i) $\lambda = \varnothing \Rightarrow \phi_{x_0} \in \Gamma_\phi\ \#\ \phi_{x_0} \in \Gamma_{<\phi>}$

(ii) $\lambda = \kappa \Rightarrow \phi_{x_0} \rightarrow^* !\phi_{x_0}\ \#\ (\phi_{x_0} \mid \{\lambda\})$ is not reduced

(iii) $\lambda \in \Gamma_\phi \Rightarrow \phi_{x_0} \rightarrow^* !\phi_{x_0}\ \#\ (\phi_{x_0} \mid \{\lambda\})$ is not reduced

(iv) $\lambda \in \Gamma_{<\phi>}\ \#\ \lambda \mid \varnothing \Rightarrow \lambda \in \Gamma_\phi \vee \lambda$ is a final value

(v) $\lambda = \tau \in (\tau_i)_i\ \#\ s \in \Gamma_\phi$

(vi) $\lambda = (\chi \rightarrow \tau) \in (\tau_i)_i\ \#\ S \in \Gamma_\phi$

To conclude the falsified assumption is always wrong, ensuring that the assumption is true.

Notice that a σ-reduction can be written $\sigma(T_i^\epsilon, T) \rightarrow_\sigma T_i^{\epsilon'}$ once reduced.

$\sigma(A, T)$: The attributes are fully instantiated here as follow:

A	$\sigma(A,T)$
$!A$	$!A$
$[A\|\tau],\ \tau \in (\tau_i)_{i \in I_n}$	$\tau' \in T\ /\ \exists j \in I_n^*,\ \exists k \in I^*_{\#T},\ t_k \in T_i^\epsilon = \tau_j \in (\tau_i)_i \wedge t'_i \in T = \tau'$ A and $\tau' \sim !A$

$(\phi_i \mid \sigma(T_i^\phi, T))_{i \in I_u}$: The semantics of inner forms and the semantics of inherited forms are quiet close for the τ-reduction.

$\forall i \in I_u,\ \phi_i \mid \sigma(T_i^\phi, T)$	$\rightarrow_x$	$\alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ !A\ (\phi_i \mid T_i^\phi)_{i \in I_u} \mid \sigma(T_i^\phi, T)$
	$\rightarrow_\tau$	$\alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ !A\ (\phi_i \mid T_i^\phi)_{i \in I_u} \mid T$
	$\rightarrow_r$	$\alpha\ (\epsilon_i \mid T_i^{\epsilon'})_{i \in I_m}\ !A\ (\phi_i \mid T_i^{\phi'})_{i \in I_u} \mid T$

2.6 Form reduction: $\phi \in \Gamma_\phi \ \rightarrow^* !\phi$

Follow the reduction of a form ϕ into its instantiated normal form $!\phi$.

$$
\begin{array}{lll}
< \phi >^+ & \rightarrow_x & < \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ !A\ (\phi_i \mid T_i^\phi)_{i \in I_u} >^+ \\
& \rightarrow_d & < \alpha\ (< \epsilon_i \mid T_i^\epsilon >^+)_{i \in I_m}\ !A\ (< \phi_i \mid T_i^\phi >^+)_{i \in I_u} >^+ \\
& \rightarrow_r^+ & < \alpha\ (!\epsilon_i)_{i \in I_m}\ !A\ (!\phi_i)_{i \in I_u} >^+ \\
& \rightarrow_\epsilon & [\alpha \mid (!\epsilon_i)_{i \in I_m}] \varnothing [!A \mid (!\epsilon_i)_{i \in I_m}]\ [(!\phi_i)_{i \in I_u} \mid (!\epsilon_i)_{i \in I_m}] \\
& \rightarrow_r & \alpha' \varnothing\ !A'\ (!\phi'_i)_{i \in I_{u'}} \\
& \rightarrow_r & !\phi
\end{array}
$$

2.7 Template form reduction: $< \phi > \in \Gamma_{<\phi>} \ \rightarrow^* !\phi$

Follow the reduction of a template form $\phi|T$ into its instantiated normal form $!\phi$.

$$
\begin{array}{lll}
< \phi \mid T >^+ & \rightarrow_x & < (\tau_i)_{i \in I_n}\ \alpha\ (\epsilon_i \mid T_i^\epsilon)_{i \in I_m}\ A\ (\phi_i \mid T_i^\phi)_{i \in I_u} \mid T >^+ \\
& \rightarrow_\tau & < \alpha\ (\epsilon_i \mid \sigma(T_i^\epsilon, T))_{i \in I_m}\ \sigma(A, T)\ (\phi_i \mid \sigma(T_i^\phi, T))_{i \in I_u} >^+ \\
& \rightarrow_\sigma & < \alpha\ (\epsilon_i \mid T_i^{\epsilon'})_{i \in I_m}\ !A\ (\phi_i \mid T_i^{\phi'})_{i \in I_u} >^+ \\
& \rightarrow_d & < \alpha\ (< \epsilon_i \mid T_i^{\epsilon'} >^+)_{i \in I_m}\ !A\ (< \phi_i \mid T_i^{\phi'} >^+)_{i \in I_u} >^+, \text{ this reduction has already been performed} \\
& \rightarrow_r^+ & < \alpha\ (!\epsilon_i)_{i \in I_m}\ !A\ (!\phi_i)_{i \in I_u} >^+ \\
& \rightarrow_\epsilon & [\alpha \mid (!\epsilon_i)_{i \in I_m}] \varnothing [!A \mid (!\epsilon_i)_{i \in I_m}]\ [(!\phi_i)_{i \in I_u} \mid (!\epsilon_i)_{i \in I_m}] \\
& \rightarrow_r & \alpha' \varnothing\ !A'\ (!\phi'_i)_{i \in I_{u'}} \\
& \rightarrow_r & !\phi
\end{array}
$$

2.8 Example

Be the product $\mathfrak{P}_{Foo} = (\Gamma,\ Foo),\ Foo \in \Gamma_\phi \subseteq \Gamma \subset \Phi$ with

- $\Gamma_\phi = \{\ Foo,\ RegWidth,\ RESERVED\ \}$
- $\Gamma_{<\phi>} = \{\ MAJOR,\ MINOR,\ REV\ \}$

2.8.1 Declaration of the set Γ

1. $Foo = all\ \varnothing\ \{\ Foo\_Revision \mid \varnothing,\ Foo\_Config \mid \varnothing\ \}$
   1. $Foo\_Revision = ro\ \{\ REV \mid \{\ [\ MAJOR \mid 0xB0\ ],\ [\ MINOR \mid 0xB1\ ]\ \}\ !o_{Foo\_Revision}\ \varnothing$
      i. $!o_{Foo\_Revision} = 0x0$
      ii. $\Gamma^{Revision} = \varnothing$
   2. $Foo\_Config = rw\ \{\ RegWidth \mid \varnothing\}\ !o_{Config}\ !\delta_{Config}\ \Gamma^{Config}$
      1. $!o_{Config} = 22 + 4 - 2$
      2. $!\delta_{Config} = \text{“}Configuration\ Register''$
      3. $Foo\_Config\_Revision = rw\ \{\ REV \mid \{\ [\ MAJOR \mid 0xA0\ ],\ [\ MINOR \mid 0xA1\ ]\ \}\ \Gamma^{Foo\_Config\_Revision}$
         i. $Foo\_Config\_Revision\_Major = ro\ \varnothing\ !\delta_{Foo\_Config\_Revision\_Major}\ \varnothing$
            - $!\delta_{Foo\_Config\_Revision\_Major} = \text{“}Major\ Number\ can\ be\ set\ by\ the\ ARM''$
            - $\Gamma^{Foo\_Config\_Revision\_Major} = \varnothing$
         ii. $\Gamma^{Foo\_Config\_Revision} = \{\ Foo\_Config\_Revision\_Major \mid \varnothing\ \}$
      4. $\Gamma^{Congig} = \{\ Foo\_Config\_Revision \mid \varnothing\ \}$
   3. $\Gamma^{Foo} = \{\ Foo\_Revision \mid \varnothing,\ Foo\_Config \mid \varnothing\ \}$
2. $MAJOR = \{\ RESET\ \}\ all\ \varnothing\ !\delta_{MAJOR}\ !\omega_{MAJOR}\ [\rho \mid RESET]\ \varnothing$
   1. $!\delta_{MAJOR} = \text{“}Major\ Revision\ Number''$
   2. $!\omega_{MAJOR} = 4\ bit$
   3. $\Gamma^{MAJOR} = \varnothing$
3. $MINOR = \{\ RESET\ \}\ all\ \varnothing\ !\delta_{MINOR}\ !\omega_{MINOR}\ [\rho \mid RESET]\ \varnothing$
   1. $!\delta_{MINOR} = \text{“}Minor\ Revision\ Number''$
   2. $!\omega_{MINOR} = 4\ bit$
   3. $\Gamma^{MINOR} = \varnothing$
4. $REV = \{\ T,\ U\ \}\ all\ \{\ RegWidth \mid \varnothing\}\ !\delta_{REV}\ \Gamma^{REV}$
   1. $!\delta_{REV} = \text{“}Revision\ Number''$
   2. $REV\_reserved = all\ \{\ RESERVED \mid \varnothing\ \}\ \varnothing$
   3. $REV\_Major = all\ \{\ T \mid \varnothing\ \}\ !o_{REV\_Major}\ \varnothing$
      i. $!o_{REV\_Major} = 4\ bit$
      ii. $\Gamma^{REV\_Major} = \varnothing$
   4. $REV\_Minor = all\ \{\ U \mid \varnothing\ \}\ !o_{REV\_Minor}\ \varnothing$
      i. $!o_{REV\_Minor} = 0x0$
      ii. $\Gamma^{REV\_Minor} = \varnothing$
   5. $\Gamma^{REV} = \{\ REV\_reserved \mid \varnothing,\ REV\_Major \mid \{\ T\ \},\ REV\_Minor \mid \{\ U\ \}\ \}$

5. $RegWidth\ =\ all\ \varnothing\ !\omega_{RegWidth}\ \varnothing$
    1. $!\omega_{RegWidth}\ =\ 32\ bit$
    2. $\Gamma^{RegWidth}\ =\ \varnothing$
6. $RESERVED\ =\ all\ \varnothing\ !\delta_{RESERVED}\ !\rho_{RESERVED}\ \varnothing$
    1. $!\delta_{RESERVED}\ =\ \text{“}Reserved''$
    2. $!\rho_{RESERVED}\ =\ 0x0$
    3. $\Gamma^{RESERVED}\ =\ \varnothing$

We therefore want to reduce Foo into a reducted normal form $!Foo$: $< \ Foo \ >^+ \rightarrow^* !Foo$, applying the rules mentioned in the sections above.

2.8.2 $< \ MAJOR \mid \kappa \ >^+ \rightarrow^* !MAJOR_\kappa, \ MAJOR \in \Gamma_{<\phi>}, \ \kappa \in \{ \ 0xB0, \ 0xA0 \ }$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{!MAJOR_\kappa}{all \varnothing \ !\delta_{MAJOR} \ !\omega_{MAJOR} \ !\rho_{MAJOR} \varnothing, \ !\rho_{MAJOR} \ = \ \kappa}}{< \ all \varnothing \ \sigma(!\delta_{MAJOR}, \ \kappa) \ \sigma(!\omega_{MAJOR}, \ \kappa) \ \sigma([\rho \mid RESET], \ \kappa) \varnothing \ >^+}}{< \ \{ \ RESET \ } all \varnothing \ !\delta_{MAJOR} \ !\omega_{MAJOR} \ [\rho \mid RESET] \varnothing \mid \kappa \ >^+}}{< \ MAJOR \mid \kappa \ >^+}$$

2.8.3 $< \ MINOR \mid \kappa \ >^+ \rightarrow^* !MINOR_\kappa, \ MINOR \in \Gamma_{<\phi>}, \ \kappa \in \{ \ 0xB1, \ 0xA1 \ }$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{!MINOR_\kappa}{\{ \ all \varnothing \ !\delta_{MINOR} \ !\omega_{MINOR} \ !\rho_{MINOR} \varnothing, \ !\rho_{MINOR} \ = \ \kappa}}{< \ \{ \ all \varnothing \ \sigma(!\delta_{MINOR}, \ \kappa) \ \sigma(!\omega_{MINOR}, \ \kappa) \ \sigma([\rho \mid RESET], \ \kappa) \varnothing \ >^+}}{< \ \{ \ RESET \ } all \varnothing \ !\delta_{MINOR} \ !\omega_{MINOR} \ [\rho \mid RESET] \varnothing \mid \kappa \ >^+}}{< \ MINOR \mid \kappa \ >^+}$$

2.8.4 $< \ RegWidth \ >^+ \rightarrow^* !RegWidth$

$$\dfrac{\dfrac{!RegWidth}{< \ all \varnothing \ !\omega_{RegWidth} \varnothing \ >^+}}{< \ RegWidth \ >^+}$$

2.8.5 $< \ RESERVED \ >^+ \rightarrow^* !RESERVED$

$$\dfrac{\dfrac{!RESERVED}{< \ all \varnothing \ !\delta_{RESERVED} \ !\rho_{RESERVED} \varnothing \ >^+}}{< \ RESERVED \ >^+}$$

2.8.6 $< \ REV \mid \{ \ \phi_1, \ \phi_2 \ } \ >^+ \rightarrow^* !REV_{inst}, \ REV \in \Gamma_{<\phi>}$

- $\phi_1 \in \{ \ MAJOR \mid \kappa \ }, \ MAJOR \in \Gamma_{<\phi>}, \ \kappa \in \{ \ 0xB0, \ 0xB1 \ }$
- $\phi_2 \in \{ \ MINOR \mid \kappa' \ }, \ MINOR \in \Gamma_{<\phi>}, \ \kappa' \in \{ \ 0xA0, \ 0xA1 \ }$
- $!\phi_1 \in \{ \ !MAJOR_{0xB0}, \ !MAJOR_{0xB1} \ }$
- $!\phi_2 \in \{ \ !MINOR_{0xA0}, \ !MINOR_{0xA1} \ }$

The table 1 depicts the reduction of the template form REV into a fully instantiated reducted normal form $!REV_{inst}$ with $inst$ an identifier highlighting the instance.

2.8.7 $< \ Foo \ >^+ \rightarrow^* !Foo$

The table 2 highlights the reduction of Foo into a reducted normal form $!Foo$.

$$\begin{array}{c}
!REV_Major_{inst},\ inst \in \{0xB0,\ 0xB1\} \\ \hline
all\ \varnothing\ !o_{REV_Major}\ !\delta_{MAJOR}\ !\omega_{MAJOR}\ !\rho_{MAJOR}\ \varnothing \\ \hline
[all \mid \{\ !\phi_1\ \}]\ \varnothing\ [!o_{REV_Major} \mid \{\ !\phi_1\ \}]\ [\varnothing \mid \{\ !\phi_1\ \}] \\ \hline
<\ all\ \{\ !\phi_1\ \}\ !o_{REV_Major}\ \varnothing\ >^+,\ !\phi_1\ =\ all\ \varnothing\ !\delta_{MAJOR}\ !\omega_{MAJOR}\ !\rho_{MAJOR}\ \varnothing \\ \hline
<\ all\ \{\ \sigma(T \mid \varnothing,\ !\phi_1)\ \}\ \sigma(!o_{REV_Major},\ !\phi_1)\ \sigma(\varnothing,\ !\phi_1)\ >^+ \\ \hline
<\ all\ \{\ T \mid \varnothing\ \}\ !o_{REV_Major}\ \varnothing \mid !\phi_1\ >^+ \\ \hline
<\ REV_Major \mid !\phi_1\ >^+
\end{array}$$

$$\begin{array}{c}
!REV_Minor_{inst},\ inst \in \{0xA0,\ 0xA1\} \\ \hline
all\ \varnothing\ !o_{REV_Minor}\ !\delta_{MINOR}\ !\omega_{MINOR}\ !\rho_{MINOR}\ \varnothing \\ \hline
[all \mid \{\ !\phi_2\ \}]\ \varnothing\ [!o_{REV_Minor} \mid \{\ !\phi_2\ \}]\ [\varnothing \mid \{\ !\phi_2\ \}] \\ \hline
<\ all\ \{\ !\phi_2\ \}\ !o_{REV_Minor}\ \varnothing\ >^+,\ !\phi_2\ =\ all\ \varnothing\ !\delta_{MINOR}\ !\omega_{MINOR}\ !\rho_{MINOR}\ \varnothing \\ \hline
<\ all\ \{\ \sigma(T \mid \varnothing,\ !\phi_2)\ \}\ \sigma(!o_{REV_Minor},\ !\phi_2)\ \sigma(\varnothing,\ !\phi_2)\ >^+ \\ \hline
<\ all\ \{\ T \mid \varnothing\ \}\ !o_{REV_Minor}\ \varnothing \mid !\phi_2\ >^+ \\ \hline
<\ REV_Minor \mid !\phi_2\ >^+
\end{array}$$

$$\begin{array}{c}
!REV_{\kappa_1,\ \kappa_2} \\ \hline
all\ \varnothing\ !\delta_{REV}\ !\omega_{RegWidth}\ \Gamma^{!REV},\ \Gamma^{!REV}\ =\ \{\ !REV_reserved,\ !REV_Major_{\kappa_1},\ !REV_Minor_{\kappa_2}\ \} \\ \hline
<\ [all \mid !RegWidth]\ \varnothing\ [!\delta_{REV} \mid !RegWidth]\ [\{\ !REV_reserved,\ !REV_Major_{\kappa_1},\ !REV_Minor_{\kappa_2}\} \mid !RegWidth]\ >^+ \\ \hline
<\ all\ \{\ !RegWidth\}\ !\delta_{REV}\ \{\ !REV_reserved,\ !REV_Major_{\kappa_1},\ !REV_Minor_{\kappa_2}\}\ >^+ \\ \hline
<\ all\ \{\ <\ RegWidth \mid \varnothing\ >^+\}\ !\delta_{REV}\ \ \{<\ REV_reserved \mid \varnothing\ >^+,\ <\ REV_Major \mid !\phi_1\ >^+,\ <\ REV_Minor \mid !\phi_2\ >^+\}\ >^+ \\ \hline
<\ all\ \{\ RegWidth \mid \varnothing\}\ !\delta_{REV}\ \ \{\ REV_reserved \mid \varnothing,\ REV_Major \mid !\phi_1,\ REV_Minor \mid !\phi_2\}\ >^+ \\ \hline
<all\ \{\ \sigma(RegWidth|\varnothing,\{\phi_1,\phi_2\})\ \}\ \sigma(!\delta_{REV},\{\phi_1,\phi_2\})\ \{\ \sigma(REV_reserved|\varnothing,\{\phi_1,\phi_2\})\ ,\sigma(REV_Major|\{T\},\{\phi_1,\phi_2\})\ ,\sigma(REV_Minor|\{U\},\{\phi_1,\phi_2\})\ \}\ >^+ \\ \hline
<\{T,U\}all\{RegWidth|\varnothing\}!\delta_{REV}\{REV_reserved|\varnothing,REV_Major|\{T\},REV_Minor|\{U\}\}|\{\phi_1,\phi_2\}>^+ \\ \hline
<\ REV \mid \{\ \phi_1,\ \phi_2\ \}\ >^+
\end{array}$$

Table 1. $<\ REV \mid \{\ \phi_1,\ \phi_2\ \}\ >^+ \longrightarrow^* !REV_{inst},\ REV \in \Gamma_{<\phi>}$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
!Foo\_Revision
}{
ro \varnothing\ !o_{Foo\_Revision}\ !\delta_{REV}\ !\omega_{RegWidth}\ \{\ !REV\_reserved,\ !REV\_Major_{0xB0},\ !REV\_Minor_{0xB1}\ \}
}
}{
< [\ ro \mid \{!REV_{0xB0,\ 0xB1}\}\ ] \varnothing\ [\ !o_{Foo\_Revision} \mid \{!REV_{0xB0,\ 0xB1}\}\ ]\ [\ \varnothing \mid \{!REV_{0xB0,\ 0xB1}\}\ ]\ >^{+}
}
}{
< ro\ \{\ !REV_{0xB0,\ 0xB1}\ \}\ !o_{Foo\_Revision}\ \varnothing\ >^{+}
}
}{
< ro\ \{ <\ REV \mid \{\ [\ MAJOR \mid 0xB0\ ],\ [\ MINOR \mid 0xB1\ ]\ >^{+}\ \}\ !o_{Foo\_Revision}\ <\ \varnothing\ >^{+}\ >^{+}
}
}{
< ro\ \{\ REV \mid \{\ [\ MAJOR \mid 0xB0\ ],\ [\ MINOR \mid 0xB1\ ]\ \}\ !o_{Foo\_Revision}\ \varnothing\ >^{+}
}
}{
< Foo\_Revision\ >^{+}
}
$$

$$
\dfrac{
\dfrac{
\dfrac{
!Foo\_Config\_Revision\_Major
}{
< ro\ \varnothing\ !\delta_{Foo\_Config\_Revision\_Major}\ \varnothing\ >^{+}
}
}{
< Foo\_Config\_Revision\_Major\ >^{+}
}
$$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
!Foo\_Config\_Revision
}{
rw\ \varnothing\ !\delta_{REV}\ !\omega_{RegWidth}\ \{\ !REV\_reserved,\ !Foo\_Config\_Revision\_Major_{0xA0},\ !REV\_Minor_{0xA1}\ \}
}
}{
!Foo\_Config\_Revision\_Major_{0xA0}\ =\ wo\ \varnothing\ !\delta_{Foo\_Config\_Revision\_Major}\ !o_{REV\_Major}\ !\omega_{MAJOR}\ !\rho_{MAJOR}\ \varnothing,\ !\rho_{MAJOR}\ =\ 0xA0
}
}{
rw\ \varnothing\ !\delta_{REV}\ !\omega_{RegWidth}\ \{\ !REV\_reserved,\ \mu[!Foo\_Config\_Revision\_Major|!REV\_Major_{0xA0}],\ !REV\_Minor_{0xA1}\ \}
}
}{
[\ rw \mid \{\ !REV_{0xA0,\ 0xA1}\ \}]\ \varnothing\ [\ \varnothing \mid \{\ !REV_{0xA0,\ 0xA1}\ \}]\ [\ \{\ !Foo\_Config\_Revision\_Major\ \} \mid \{\ !REV_{0xA0,\ 0xA1}\ \}]
}
}{
< rw\ \{\ !REV_{0xA0,\ 0xA1}\ \}\ \{\ !Foo\_Config\_Revision\_Major\ \}\ >^{+}
}
}{
< rw\ \{ <\ REV \mid \{\ [\ MAJOR \mid 0xA0\ ],\ [\ MINOR \mid 0xA1\ ]\ >^{+}\ \}\ \{ <\ Foo\_Config\_Revision\_Major \mid \varnothing\ >^{+}\ \}\ >^{+}
}
}{
< rw\ \{\ REV \mid \{\ [\ MAJOR \mid 0xA0\ ],\ [\ MINOR \mid 0xA1\ ]\ \}\ \{\ Foo\_Config\_Revision\_Major \mid \varnothing\ \}\ >^{+}
}
}{
< Foo\_Config\_Revision\ >^{+}
}
$$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
!Foo\_Config
}{
rw\ \varnothing\ !o_{Config}\ !\delta_{Config}\ \omega_{RegWidth}\ \{\ !Foo\_Config\_Revision\}
}
}{
[\ rw \mid \{!RegWidth\}]\ \varnothing\ [\ !o_{Config}\ !\delta_{Config} \mid \{!RegWidth\}]\ [\ \{!Foo\_Config\_Revision\} \mid \{!RegWidth\}]
}
}{
< rw\ \{!RegWidth\}\ !o_{Config}\ !\delta_{Config}\ \{!Foo\_Config\_Revision\}\ >^{+}
}
}{
< rw\ \{<\ RegWidth \mid \varnothing\ >^{+}\}\ !o_{Config}\ !\delta_{Config}\ \{\ <\ Foo\_Config\_Revision \mid \varnothing\ >^{+}\ \}\ >^{+}
}
}{
< rw\ \{RegWidth \mid \varnothing\}\ !o_{Config}\ !\delta_{Config}\ \{\ Foo\_Config\_Revision \mid \varnothing\ \}\ >^{+}
}
}{
< Foo\_Config\ >^{+}
}
$$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
!Foo
}{
all\ \varnothing\ \{\ !Foo\_Revision,\ !Foo\_Config\ \}
}
}{
all\ \varnothing\ \{\ <\ Foo\_Revision\ >^{+},\ <\ Foo\_Config\ >^{+}\ \}
}
}{
< all\ \varnothing\ \{\ Foo\_Revision \mid \varnothing,\ Foo\_Config \mid \varnothing\ \}\ >^{+}
}
}{
< Foo\ >^{+}
}
$$

Table 2. $< Foo >^{+} \rightarrow^{*} !Foo$

3 Language Definition

The language proposal RD[1] aims to fit the architecture and the MoC described in the previous chapter.

3.1 EBNF Grammar

3.1.1 Expressions

expr	::=	expr (’+’ | ’-’ | ’*’ | ’/’ | ’%’) expr | ’(’ expr ’)’ | VALUE
IDT	::=	[a-zA-Z]([a-zA-Z0-9] | ’_’)*
VALUE	::=	BIN_VALUE | HEX_VALUE | DEC_VALUE
BIN_VALUE	::=	’0’ [by] (0 | 1)+
HEX_VALUE	::=	’0’ [x] [0-9a-fA-F]+
DEC_VALUE	::=	[0-9]+
texpr	::=	texpr (’+’ | ’-’ | ’*’ | ’/’ | ’%’) texpr | ’(texpr ’)’ | VALUE | IDT

The operator and their precedence are defined as follow.

operator	type	precedence	comment
’+’	binary	left	plus
’+’	hexadecimal	left	plus
’+’	decimal	left	plus
’-’	binary	left	minus
’-’	hexadecimal	left	minus
’-’	decimal	left	minus
’*’	binary	left	*not defined*
’*’	hexadecimal	left	*not defined*
’*’	decimal	left	multiplication without overload
’/’	binary	left	*not defined*
’/’	hexadecimal	left	*not defined*
’/’	decimal	left	integer division
’%’	binary	left	*not defined*
’%’	hexadecimal	left	*not defined*
’%’	decimal	left	modulus

3.1.2 Form

form	::=	access ID extends (“is” body)? “end” ID ’;’
access	::=	“all” | “ro” | “wo” | “rw”
extends	::=	“extends” extends_form (’,’ extends_form)* | ϵ
extends_form	::=	ID | ID ’<’ IDV (’,’ IDV)* ’>’
IDV	::=	ID | UNIT | VALUE | STRING
ID	::=	[a-zA-Z]([a-zA-Z0-9] | ’_’)*
UNIT	::=	“bit” | “byte”
body	::=	(offset | width | description | reset | inner_form)+
offset	::=	“offset” (“init” | expr UNIT | expr) ’;’
width	::=	“width” expr UNIT ’;’
description	::=	“description” STRING ’;’
reset	::=	“reset” expr ’;’
STRING	::=	’“’ < printable character >* ’“’

[1] RD as **R**egister **D**escription

3.1.3 Template Form

tform	::=	“template” template access ID extends (“is” body)? “end” ID ’;’
template	::=	’<’ IDT (’,’ IDT)* ’>’
access	::=	“all” | “ro” | “wo” | “rw”
extends	::=	“extends” extends_form (’,’ extends_form)* | ϵ
extends_form	::=	ID | IDT | ID ’<’ IDV (’,’ IDV)* ’>’
IDV	::=	ID | UNIT | VALUE | STRING | IDT
UNIT	::=	“bit” | “byte” | IDT
body	::=	(offset | width | description | reset | inner_form)+
offset	::=	“offset” (“init” | texpr UNIT | texpr) ’;’
width	::=	“width” texpr UNIT ’;’
description	::=	“description” (STRING | IDT)’;’
reset	::=	“reset” texpr ’;’
STRING	::=	’“’ < printable character >* ’“’

3.1.4 Inner Form

inner_form	::=	access [ID | “reserved] extends (“is” body)? “end” [ID | “reserved”] ’;’
access	::=	“all” | “ro” | “wo” | “rw”
extends	::=	“extends” extends_form (’,’ extends_form)* | ϵ
extends_form	::=	ID | IDT | ID ’<’ IDV (’,’ IDV)* ’>’
IDV	::=	ID | UNIT | VALUE | STRING | IDT
UNIT	::=	“bit” | “byte” | IDT
body	::=	(offset | width | description | reset | inner_form)+ | “mandatory”
offset	::=	“offset” (“init” | texpr UNIT | texpr) ’;’
width	::=	“width” texpr UNIT ’;’
description	::=	“description” (STRING | IDT)’;’
reset	::=	“reset” texpr ’;’
STRING	::=	’“’ < printable character >* ’“’

3.1.5 Product

A product $\mathfrak{P}$ is composed of a set of forms Γ_ϕ and a set of template forms $\Gamma_{<\phi>}$ to define the context Γ and a root node $\nu \in \Gamma_\phi$.

root	::=	“root” form

3.2 Modular Compiling

Compiling a form $\phi \in \Gamma_\phi$ to reduce it into $!\phi$ costs time to perform the checks. Besides, a compiling costs memory to fully instantiate ϕ into $!\phi$. Unfortunately a root node can be derivated in too much levels in depth and width to manage a fully instantiation, costing by side effect too much time and/or too much memory leading to system issues and of course cause system abortion befor getting $!\phi$.

To manage wide node instantiation, the level of depth and width can be reduced, compiling nodes, which can be, as separate entities in order to perform modular compiling.

Be $\mathfrak{P} = (\Gamma, \phi)$ so that we want to compute ϕ into $!\phi$. Unfortunately, this product has scalable issues due to the size of its derivated tree.

We can however observe that $\phi \ \rightarrow^n \ \phi^n \ \rightarrow^m !\phi$ where ϕ^n is the n^{th}-derivate from ϕ. Moreover, $\psi \in \Gamma$ needs to be inherited within ϕ^n. Thus, we can campile ψ as a separate root node leading to $!\psi \in \Phi$ a fully instantiated form. And then, substitute in Γ_ϕ, ψ by $!\psi$.

To conclude, to compute ϕ, we've just given a way to cut the computation taking ψ as the root of a sub-tree needed to get $!\phi$ fully instantiated. We've therefore gained time and memory in the computation of ϕ, using a precompiled form $!\psi$ instead of ψ which is already fully instantiated.

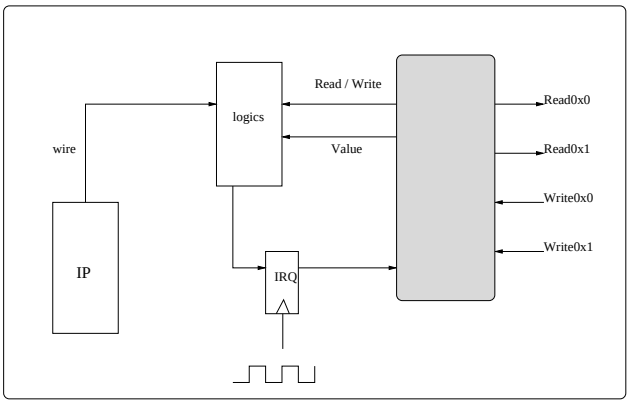
Cutting such sub-trees hierarchically may lead to a computation of ϕ which may cost neither time nor memory, whereas with the first approach ϕ has not been previously computed due to the explosion in time and/or memory of its huge tree.

4 Example of an IRQ management module

An IRQ management module is composed of IRQ lines which will be enabled or disabled by hardware components. These lines can be read to get if the concerned IRQ has been arisen and reset according to 4 commonly used protocols[2].

For our purpuse, the truth table below depicts the behavior of a read / write 1 to clear IRQ line.

In this example, we assume that no bypass protocol is defined i.e. an IRQ arisen by a component and a read/write command cannot occur at the meantime. The figure below highlights a basic IRQ line management.



wire	IRQ	command	IRQ
0	0	idle	0
0	1	idle	0
1	0	idle	1
1	1	idle	1
0	0	read	0 (Read0x0)
0	0	write 1 (Write0x1)	0
0	0	write 0)Write0x0)	0
0	1	read	1 (Read0x1)
0	1	write 1 (Write0x1)	0
0	1	write 0 (Write0x0)	1

4.1 Generic read / write 1 to clear form declaration

This form is basically fully instantiated yet:

```
rw IRQ_rw1toClr is
   description ‘‘IRQ line R/W 1 to clear’’;
   reset 0;
   width 1 bit;
   ro Read0x0 is
      description ‘‘read the value 0x0’’;
      reset 0x0;
   end Read0x0;
   ro Read0x1 is
      description ‘‘read the value 0x1’’;
      reset 0x1;
   end Read0x0;
   wo Write0x0 is
      description ‘‘writing the value 0x0 has no effet’’;
   end Read0x0;
   wo Write0x1 is
      description ‘‘write the value 0x1 to clear the register’’;
      reset 0x0;
   end Read0x0;
end IRQ_rw1toClr;
```

[2] read / write 1 to clear, read / write 0 to clear, read / write 1 to set, read / write 0 to set

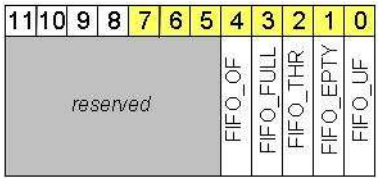
However, the reset value of this 1-bit register is hard coded with 0. In some cases, this value should be set to 1. The first solution would be to manually duplicate this capture to hard code the reet value to 1. The second solution, highly recommended, is to give a parameter to this form. This fully instantiated form in Γ_{ϕ} is abstracted and goeas in $\Gamma_{<\phi>}$ in order to be instantiated with the wanted reset value.

```
template < RESET >
rw IRQ_rw1toClr is
   description ‘‘IRQ line R/W 1 to clear’’;
   reset RESET;
   width 1 bit;
   ro Read0x0 is
      description ‘‘read the value 0x0’’;
      reset 0x0;
   end Read0x0;
   ro Read0x1 is
      description ‘‘read the value 0x1’’;
      reset 0x1;
   end Read0x0;
   wo Write0x0 is
      description ‘‘writing the value 0x0 has no effet’’;
   end Read0x0;
   wo Write0x1 is
      description ‘‘write the value 0x1 to clear the register’’;
      reset 0x0;
   end Read0x0;
end IRQ_rw1toClr;
```

4.2 Declaration of an IRQ management module

We are designing a FIFO which has 12 IRQ lines. We will declare and define the module which aim to manage these lines.

The current specification defines 5 IRQs as depicted by the figure below.



We can notice that 7 bits are reserved for future usage if needed in the beach $[\![5,\ 11]\!]$. These bits are hard-wired connected to return 0’s on read and discard any writing values. We will therefore define a template form which declares such beach of reserved bits.

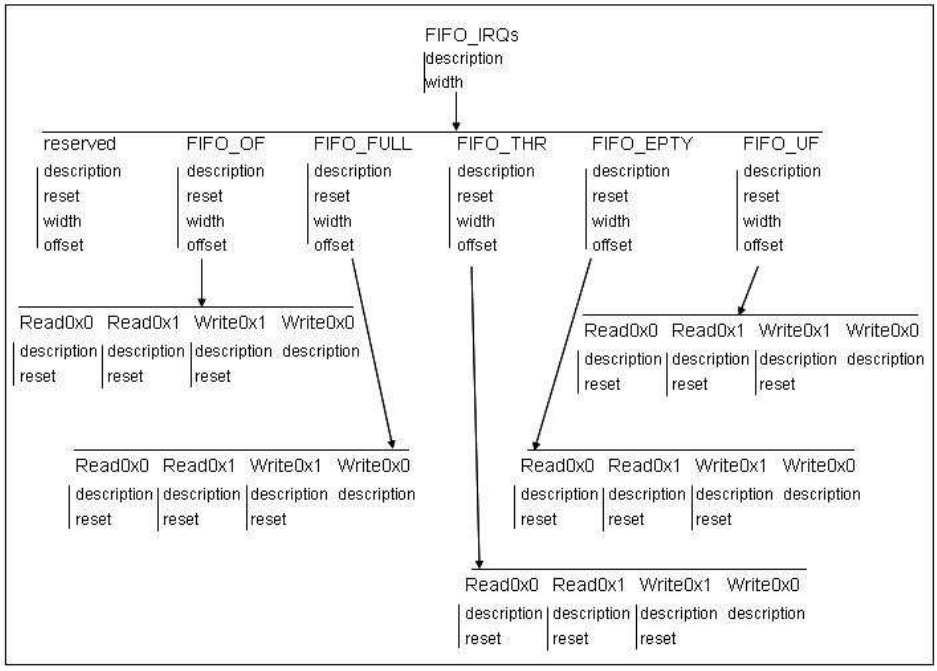
```
template < N, OFFSET >
rw Reserved is
   description ‘‘Reserved. Reads return 0’s’’;
   reset 0x0;
   width N bit;
   offset OFFSET bit;
end Reserved;

main all FIFO_IRQs is
   description ‘‘FIFO IRQ lines management’’;
   width 12 bit;
   all reserved extends Reserved< 12 - 5, 5 > end reserved;
   all FIFO_UF extends IRQ_rw1toClr<0> is
      description ‘‘FIFO UnderFlow IRQ’’;
      offset init;
   end FIFO_UF;
   all FIFO_EPTY extends IRQ_rw1toClr<1> is
      description ‘‘FIFO Empty IRQ’’;
      offset 1 bit;
   end FIFO_EPTY;
   all FIFO_THR extends IRQ_rw1toClr<0> is
      description ‘‘The threshold in the FIFO has been reached’’;
      offset 2 bit;
```

```
   end FIFO_THR;
   all FIFO_FULL extends IRQ_rw1toClr<0> is
      description ‘‘The FIFO is Full’’;
      offset 3 bit;
   end FIFO_FULL;
   all FIFO_OF extends IRQ_rw1toClr<0> is
      description ‘‘FIFO OverFlow IRQ’’;
      offset 4 bit;
   end FIFO_OF;
end FIFO_IRQs;
```

4.3 Fully instantiated IRQ management component for the FIFO

Hence, our system $\mathfrak{P}_{FIFO\_IRQs} = (\Gamma, FIFO\_IRQs)$, $\Gamma_\phi = \{ FIFO\_IRQs \}$, $\Gamma_{<\phi>} = \{ Reserved, IRQ\_rw1toClr \}$ can be computed to get reducted into $!FIFO\_IRQs$ as depicted by the following tree. Our compacted 50 lines description, is computed into a RDO outputted flatten file, 115 lines wide, fully RD compliant. Consequently, as soon as $FIFO\_IRQs \in \Gamma_\phi$ is reduced into a normal form $!FIFO\_IRQs$, we can use it as a core component avoiding costly reductions.




```
all FIFO_IRQs is
   description "FIFO IRQ lines management";
   width 12 bit;
   rw reserved is
      description "Reserved. Reads return 0’s";
      offset 5 bit;
      reset 0x0;
      width 7 bit;
   end reserved;
   rw FIFO_UF is
      description "FIFO UnderFlow IRQ";
      offset 0x0;
      reset 0;
      width 1 bit;
      ro Read0x0 is
         description "read the value 0x0";
         reset 0x0;
      end Read0x0;
      ro Read0x1 is
         description "read the value 0x1";
         reset 0x1;
```

```
   end Read0x1;
   wo Write0x0 is
      description "writing the value 0x0 has no effet";
   end Write0x0;
   wo Write0x1 is
      description "write the value 0x1 to clear the register";
      reset 0x0;
   end Write0x1;
end FIFO_UF;
rw FIFO_EPTY is
   description "FIFO Empty IRQ";
   offset 1 bit;
   reset 1;
   width 1 bit;
   ro Read0x0 is
      description "read the value 0x0";
      reset 0x0;
   end Read0x0;
   ro Read0x1 is
      description "read the value 0x1";
      reset 0x1;
   end Read0x1;
   wo Write0x0 is
      description "writing the value 0x0 has no effet";
   end Write0x0;
   wo Write0x1 is
      description "write the value 0x1 to clear the register";
      reset 0x0;
   end Write0x1;
end FIFO_EPTY;
rw FIFO_THR is
   description "The threshold in the FIFO has been reached";
   offset 2 bit;
   reset 0;
   width 1 bit;
   ro Read0x0 is
      description "read the value 0x0";
      reset 0x0;
   end Read0x0;
   ro Read0x1 is
      description "read the value 0x1";
      reset 0x1;
   end Read0x1;
   wo Write0x0 is
      description "writing the value 0x0 has no effet";
   end Write0x0;
   wo Write0x1 is
      description "write the value 0x1 to clear the register";
      reset 0x0;
   end Write0x1;
end FIFO_THR;
rw FIFO_FULL is
   description "The FIFO is Full";
   offset 3 bit;
   reset 0;
   width 1 bit;
   ro Read0x0 is
      description "read the value 0x0";
      reset 0x0;
   end Read0x0;
   ro Read0x1 is
      description "read the value 0x1";
      reset 0x1;
   end Read0x1;
   wo Write0x0 is
      description "writing the value 0x0 has no effet";
   end Write0x0;
   wo Write0x1 is
      description "write the value 0x1 to clear the register";
      reset 0x0;
   end Write0x1;
end FIFO_FULL;
rw FIFO_OF is
   description "FIFO OverFlow IRQ";
   offset 4 bit;
   reset 0;
   width 1 bit;
```

```
      ro Read0x0 is
         description "read the value 0x0";
         reset 0x0;
      end Read0x0;
      ro Read0x1 is
         description "read the value 0x1";
         reset 0x1;
      end Read0x1;
      wo Write0x0 is
         description "writing the value 0x0 has no effet";
      end Write0x0;
      wo Write0x1 is
         description "write the value 0x1 to clear the register";
         reset 0x0;
      end Write0x1;
   end FIFO_OF;
end FIFO_IRQs;
```

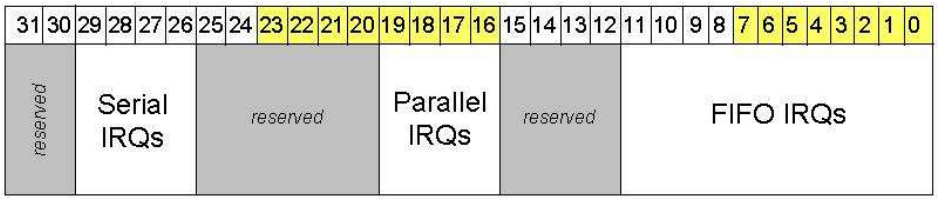
4.4 Component managing the IRQs of a module

This section aims at describe how to easily use the IRQ management component defined above as a part of this component registers description using a FIFO, but does not focus on a fully description.

Be the following piece of registers specification of the Camera Core module. All registers are 32-bit wide.

Register	Offset	Description
CC_REVISION	0x00	Revision Register
CC_IRQSTATUS	0x18	Interrupt Status Register
CC_IRQENABLE	0x1C	Interrupt Enable Register
CC_FIFODATA	0x4C	FIFO Data Register
CC_TEST	0x50	Test Register

The piece of specification below of the CC_IRQSTATUS register depicts the ranges affected for the three main sub-modules of the Camera Core IP.




```
include ‘‘common.rd’’ /*** the file includes the template form Reserved      ***/
import ‘‘FIFO_IRQs’’  /*** the fully instantiated form FIFO_IRQs is imported ***/

/*** The form should have been declared in the file common.rd ***/
all XXXXxxxxRegisterWidth is
   width 32 bit;
end XXXXxxxxRegisterWidth;

template < RESET, N, OFFSET >
ro ReservedDoNotWrite extends Reserved<N, OFFSET > is
   description ‘‘Reserved. Do Not Write’’;
   reset RESET;
end ReservedDoNotWrite;

all IRQSTATUS extends XXXXxxxxRegisterWidth is
   description ‘‘Interrupt Status Register’’;
   all FIFO extends FIFO_IRQs is
      offset init;
   end FIFO;
   all reserved_1 extends Reserved<16 - 12, 12> end reserved;
   all PARALLEL ... end PARALLEL;
   all reserved_2 extends Reserved<26 - 20, 20> end reserved;
   all SERIAL ... end SERIAL;
   all reserved_3 extends ReservedDoNotWrite<0b11, 2, 30> end reserved;
```

```
end IRQSTATUS;

main all CameraCore is
   description ‘‘Camera Core Registers’’;
   ro CC_REVISION  ... end CC_REVISION;
   rw CC_IRQSTATUS extends IRQSTATUS is
      offset 0x18;
   end CC_IRQSTATUS;
   rw CC_IRQENABLE ... end CC_IRQENABLE;
   rw CC_FIFODATA  ... end CC_FIFODATA;
   ro CC_TEST      ... end CC_TEST;
end CameraCore;
```