

Cryptocode

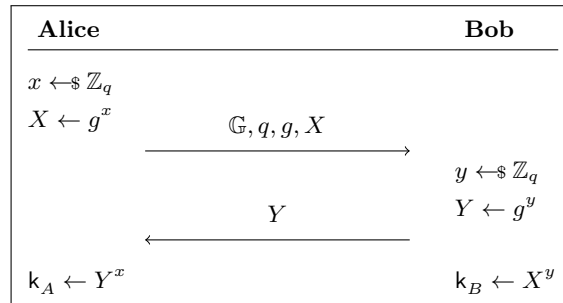
TYPESETTING CRYPTOGRAPHY
Version v0.44

Arno Mittelbach
mail@arno-mittelbach.de
<https://github.com/arnomi/cryptocode>*

August 15, 2021

Abstract

The cryptocode package provides a set of macros to ease the typesetting of pseudocode, algorithms and protocols (such as the one below). In addition it comes with a wide range of tools to typeset cryptographic papers (hence the name). This includes simple predefined commands for typesetting probabilities and “commonly encountered math” as well as for concepts such as a security parameter 1^n or advantage terms $\text{Adv}_{\mathcal{A}, \text{PRF}}^{\text{pf}}(n) = \text{negl}(n)$. Furthermore, it includes environments to layout game-based proofs or black-box reductions.



*If you use cryptocode in your work, consider starring the repository on GitHub and/or rating it on CTAN.

Contents

1	Cryptocode by Example	1
1.1	Pseudocode	1
1.2	Stacking	3
1.3	Columns	4
1.4	Protocols	5
1.5	Game-Based Proofs	6
1.6	Black-Box Reductions	6
2	Notation Macros	8
2.1	Security Parameter	8
2.2	Advantage Terms	8
2.3	Math Operators	9
2.4	Adversaries	9
2.5	Landau	10
2.6	Probabilities	10
2.7	Sets	11
2.8	Cryptographic Notions	12
2.9	Logic	12
2.10	Function Families	13
2.11	Machine Model	13
2.12	Crypto Primitives	14
2.13	Oracles	14
2.14	Events	15
2.15	Complexity	15
2.16	Asymptotics	16
2.17	Keys	16
3	Pseudocode	17
3.1	Basics	17
3.1.1	Customizing Pseudocode	18
3.1.2	Customized Pseudocode Commands	20
3.2	Indentation	21
3.3	Textmode	23
3.4	Syntax Highlighting	24
3.4.1	Automatic Syntax Highlighting (Experimental)	25
3.5	Line Numbering	28
3.5.1	Skipping Line Numbers	28
3.5.2	Manually Inserting Line Numbers	28
3.5.3	Start Values	29
3.5.4	Separators	29
3.5.5	Style	29
3.6	Subprocedures	30
3.6.1	Numbering in Subprocedures	30
3.7	Stacking Procedures	31
3.7.1	Stacking Options	33
3.8	Default Arguments	33
3.9	Divisions and Linebreaks	34
3.9.1	Optimizing Layout	35
3.10	Matrices and Math Environments within Pseudocode	35
3.11	Fancy Code with Overlays	36

4	Tabbing Mode	38
4.1	Tabbing in Detail	38
4.1.1	Overriding The Tabbing Character	39
4.1.2	Custom Line Spacing and Horizontal Rules	39
5	Protocols	40
5.1	Tabbing	42
5.2	Multiline Messages	42
5.2.1	Multiplayer Protocols	43
5.2.2	Divisions	43
5.3	Line Numbering in Protocols	44
5.3.1	Separators	45
5.3.2	Spacing	45
5.4	Sub Protocols	46
5.5	Compact Presentation of Protocols	46
6	Game-Based Proofs	48
6.1	Basics	48
6.1.1	Highlight Changes	48
6.1.2	Boxed Games	49
6.1.3	Reduction Hints	49
6.1.4	Numbering and Names	50
6.1.5	Default Name and Argument	51
6.1.6	Bi-Directional Games	51
6.1.7	Styling Game Procedures	52
6.2	Game Descriptions	52
7	Black-Box Reductions	54
7.1	Nesting of Boxes	56
7.2	Messages and Queries	57
7.2.1	Options	58
7.2.2	Vdots	60
7.2.3	Add Space	60
7.2.4	Loops	62
7.2.5	Intertext	63
7.3	Oracles	64
7.3.1	Communicating with Oracles	65
7.4	Challengers	66
7.4.1	Communicating with Challengers	67
7.5	Horizontal Stacking	68
7.6	Examples	68
8	Known Issues	71
8.1	Pseudocode KeepSpacing within Commands	71
8.2	AMSFonTS	71
8.3	Hyperref	71
8.4	Cleveref	71
8.5	Babel - Spanish	71

9	Implementation	73
9.1	Package Options	73
9.1.1	operators	73
9.1.2	adversary	74
9.1.3	landau	74
9.1.4	probability	75
9.1.5	sets	76
9.1.6	noamsfonts	76
9.1.7	notions	76
9.1.8	logic	77
9.1.9	ff (function families)	77
9.1.10	mm (machine models)	77
9.1.11	advantage	78
9.1.12	primitives	78
9.1.13	oracles	79
9.1.14	events	79
9.1.15	complexity	79
9.1.16	asymptotics	80
9.1.17	keys	80
9.1.18	Security parameter	81
9.2	Preamble and Option Parsing	81
9.3	Global Macros	82
9.3.1	Styles	82
9.3.2	Order of Growth	82
9.3.3	Spacing	82
9.3.4	Keywords and Highlighting	82
9.3.5	Misc	83
9.4	Internal Helper Functions	83
9.5	Stacking	85
9.5.1	Manual Spacing	85
9.5.2	Misc	85
9.5.3	Stacking Options	85
9.5.4	The Stacking Environments	87
9.6	The pseudocode command	88
9.6.1	Options	90
9.6.2	Automatic Syntax Highlighting and Spacing (Experimental)	92
9.6.3	Helper Variables	95
9.6.4	The Actual Pseudocode Command	95
9.7	Create Pseudocode/Procedure Commands	99
9.8	Subprocedures	100
9.9	Protocols	101
9.10	Tikz within Pseudocode	104
9.11	Black Box Reductions	105
9.12	Game-Based Proofs	118
9.12.1	Game Descriptions	122

1 Cryptocode by Example

The cryptocode package provides a set of commands to ease the typesetting of pseudocode, protocols, game-based proofs and black-box reductions. In addition it comes with a large number of predefined commands. In this section we present the various features of cryptocode by giving small examples. But first, let's load the package

```
1 \usepackage[
2   n, % or lambda
3   advantage,
4   operators,
5   sets,
6   adversary,
7   landau,
8   probability,
9   notions,
10  logic,
11  ff,
12  mm,
13  primitives,
14  events,
15  complexity,
16  oracles,
17  asymptotics,
18  keys
19 ]{cryptocode}
```

Note that all the options refer to a set of commands. That is, without any options cryptocode will provide the mechanisms for writing pseudocode, protocols, game-based proofs and black-box reductions but not define additional commands, such as `\pk` or `\sk` (for typesetting public and private/secret keys) which are part of the keys option. We discuss the various options and associated commands in Section 2.

1.1 Pseudocode

The cryptocode package tries to make writing pseudocode easy and enjoyable. The `\pseudocode` command takes a single parameter where you can start writing code in mathmode using `\\` as line breaks. Following is an IND-CPA game definition using various commands from cryptocode to ease writing keys (`\pk`, `\sk`), sampling (`\sample`), and more:

```
1 :  $b \leftarrow \{0, 1\}$ 
2 :  $(pk, sk) \leftarrow \text{KGen}(1^n)$ 
3 :  $(state, m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$ 
4 :  $c \leftarrow \text{Enc}(pk, m_b)$ 
5 :  $b' \leftarrow \mathcal{A}(1^n, pk, c, state)$ 
6 : return  $b = b'$ 
```

```
1 \pseudocode[linenumbering]{
2   b \sample \bin \\
3   (\pk, \sk) \sample \kgen (\seccparam) \\
4   (\state, m_0, m_1) \sample \adv(\seccparam, \pk, c) \\
5   c \sample \enc(\pk, m_b) \\
6   b' \sample \adv(\seccparam, \pk, c, \state) \\
7   \pcreturn b = b' }
```

In many cases, we want to set pseudocode blocks in-between paragraphs with spacing similar to how we would offset equations. For this, and for laying out multiple code blocks, cryptocode offers “stacking” environments `\pchstack` and `\pcvstack`. For typesetting a code block nicely centered and boxed

```

1 :   $b \leftarrow \{0, 1\}$ 
2 :   $(pk, sk) \leftarrow \text{KGen}(1^n)$ 
3 :   $(state, m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$ 
4 :   $c \leftarrow \text{Enc}(pk, m_b)$ 
5 :   $b' \leftarrow \mathcal{A}(1^n, pk, c, state)$ 
6 :  return  $b = b'$ 

```

you could thus use:

```

1 \begin{pchstack}[center,boxed]
2   \pseudocode[linenumbering]{
3     b \sample \bin \\
4     (\pk,\sk) \sample \kgen (\seccparam) \\
5     (\state,m_0,m_1) \sample \adv(\seccparam, \pk, c) \\
6     c \sample \enc(\pk,m_b) \\
7     b' \sample \adv(\seccparam, \pk, c, \state) \\
8     \pcreturn b = b' }
9 \end{pchstack}

```

As this is a common task, cryptocode offers the `\pseudocodeblock` command which is a shorthand for the above (without the frame). In case you want to provide different options or a shorter command (say `\pcb`) you can easily define the command via

```

1 \createpseudocodeblock{pcb}{center,boxed}{}{}

```

The above could now be written, more succinctly as

```

1 \pcb[linenumbering]{
2   b \sample \bin \\
3   (\pk,\sk) \sample \kgen (\seccparam) \\
4   (\state,m_0,m_1) \sample \adv(\seccparam, \pk, c) \\
5   c \sample \enc(\pk,m_b) \\
6   b' \sample \adv(\seccparam, \pk, c, \state) \\
7   \pcreturn b = b'
8 }

```

The pseudocode command (and its block variant) takes a single mandatory argument (the code) plus an optional argument which allows you to specify options in a key=value fashion. In the above example we used the `linenumbering` option.

It is easy to define a heading for your code. Either specify the header using the option “head” or use the `\procedure` command (or its block variant `\procedureblock`) which takes an additional argument to specify the headline.

$$\text{IND-CPA}_{\text{Enc}}^A(n)$$

```

1 :   $b \leftarrow \{0, 1\}$ 
2 :   $(pk, sk) \leftarrow \text{KGen}(1^n)$ 
3 :   $(state, m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$ 
4 :   $c \leftarrow \text{Enc}(pk, m_b)$ 
5 :   $b' \leftarrow \mathcal{A}(1^n, pk, c, state)$ 
6 :  return  $b = b'$ 

```

```

1 \procedureblock[linenumbering]{\indcpa_\enc^adv(\secpa)}{
2   b \sample \bin \\\
3   (\pk,\sk) \sample \kgen(\secpa) \\\
4   (\state,m_0,m_1) \sample \adv(\secpa, \pk, c) \\\
5   c \sample \enc(\pk,m_b) \\\
6   b' \sample \adv(\secpa, \pk, c, \state) \\\
7   \pcreturn b = b' }

```

Similarly to before, we can define a shorthand and boxed variant as

```

1 \createprocedureblock{procb}{center,boxed}{}{}{}

```

There is a lot more that we will discuss in detail in Section 3. Here, for example, is the same code with an overlay explanation and a division of the pseudocode.

IND-CPA_{Enc}^A(n)

```

1 : b ←$ {0, 1}
2 : (pk, sk) ←$ KGen(1n)
... Setup Completed ...
3 : (m0, m1) ←$ A(1n, pk, c)
4 : c ←$ Enc(pk, mb)
5 : b' ←$ A(1n, pk, c, state)
6 : return b = b'

```

KGen(1ⁿ) samples a public key pk and a private key sk.

```

1 \begin{pimage}
2 \procedureblock[linenumbering]{\indcpa_\enc^adv(\secpa)}{\%
3   b \sample \bin \\\
4   (\pk,\sk) \sample \kgen(\secpa)\pcnode{kgen} \pclb
5   \pcintertext[dotted]{Setup Completed}
6   (m_0,m_1) \sample \adv(\secpa, \pk, c) \\\
7   c \sample \enc(\pk,m_b) \\\
8   b' \sample \adv(\secpa, \pk, c, \state) \\\
9   \pcreturn b = b' }
10
11 \pcdraw{
12   \node[rectangle callout, callout absolute pointer=(kgen), fill=orange]
13     at ([shift={(+3,-1)}]kgen) {
14       \begin{varwidth}{3cm}
15         $\kgen(\secpa)$ samples a public key $\pk$ and a private key $\sk$.
16       \end{varwidth}
17     };
18   }
19 \end{pimage}

```

1.2 Stacking

To arrange multiple procedures, cryptocode offers horizontal and vertical stacking environments `\pchstack` and `\pcvstack`. In the example below we arrange four code blocks in three columns equispaced with 1cm distance and stack two procedures in the center column.

A	B.1	C
$a \leftarrow \$ A$	$b \leftarrow \$ B$	$c \leftarrow \$ C$
return a	return b	return c
	B.1	
	$b \leftarrow \$ B$	
	return b	

```

1 \begin{pchstack}[center,space=1cm]
2   \procedure{A}{
3     a \sample A \\
4     \pcreturn a
5   }
6
7   \begin{pcvstack}[boxed,space=0.5cm]
8     \procedure{B.1}{
9       b \sample B \\
10      \pcreturn b
11    }
12    \procedure{B.1}{
13      b \sample B \\
14      \pcreturn b
15    }
16  \end{pcvstack}
17
18  \procedure{C}{
19    c \sample C \\
20    \pcreturn c
21  }
22 \end{pchstack}

```

1.3 Columns

The `\pseudocode` and `\procedure` commands allow the usage of multiple columns. You switch to a new column by inserting a `\>`. This is similar to using an `align` environment and placing a tabbing & character.¹

First	Second	Third	Fourth
$b \leftarrow \$ \{0, 1\}$	$b \leftarrow \$ \{0, 1\}$	$b \leftarrow \$ \{0, 1\}$	$b \leftarrow \$ \{0, 1\}$

```

1 \pseudocodeblock{%
2   \textbf{First} \> \textbf{Second} \> \textbf{Third} \> \textbf{Fourth} \\
3   b \sample \bin \> b \sample \bin \> b \sample \bin \> b \sample \bin}

```

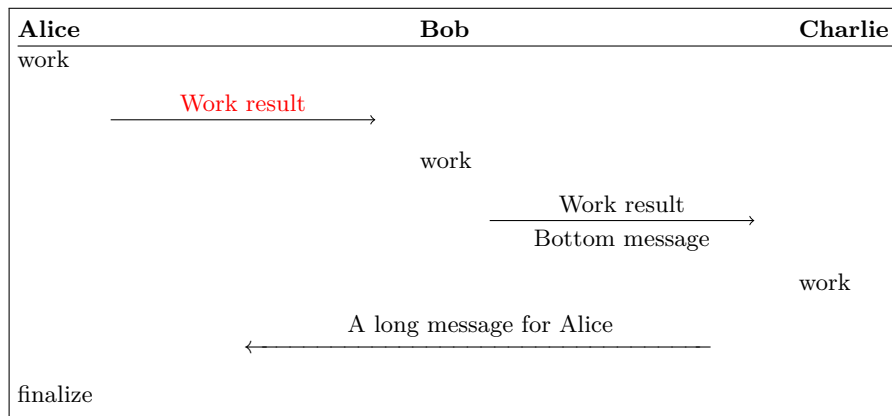
As you can see the first column is left aligned the second right, the third left and so forth. In order to get only left aligned columns you could thus always skip a column by using `\>\>` or you can alternatively use `\<` as a shorthand for `\>\>`.

First	Second	Third	Fourth
$b \leftarrow \$ \{0, 1\}$	$b \leftarrow \$ \{0, 1\}$	$b \leftarrow \$ \{0, 1\}$	$b \leftarrow \$ \{0, 1\}$

¹In fact, the *pseudocode* command is based on `amsmath`'s `flalign` environment.

1.4 Protocols

Using columns makes it easy to write even complex protocols. Following is a simple three-party protocol.



```

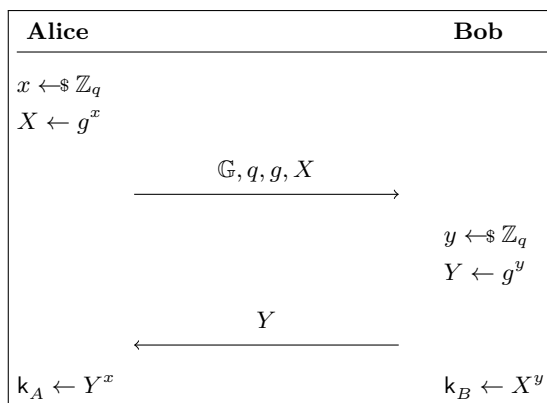
1 \psudocodeblock{
2   \textbf{Alice} < < \textbf{Bob} < < \textbf{Charlie} \\[[\hline]
3   \text{work} < < < < \\
4   < \sendmessageright{top=Work result ,topstyle=red} < < < \\
5   < < \text{work} < < \\
6   < < < \sendmessageright{top=Work result ,bottom=Bottom message} < \\
7   < < < < \text{work} \\
8   < \sendmessageleftx{8}{\text{A long message for Alice}} < \\
9   \text{finalize} < < < < }

```

The commands `\sendmessageright` and `\sendmessageleft` are very flexible and allow to style the sending of messages in various ways. Also note the `\\[\\hline]` at the end of the first line. Here the first optional argument allows us to specify the lineheight (similarly to the behavior in an `align` environment) while the second optional argument allows us to, for example, draw a horizontal line.

In multi-player protocols such as the one above the commands `\sendmessagerightx` and `\sendmessageleftx` (note the x at the end) allow to send messages over multiple columns. In the example, as we were using `\c` the final message thus spans 8 columns.

For basic protocols you might also utilize the `\sendmessageright*` and `\sendmessageleft*` commands which simply take a message which is displayed (in math mode) on top.



```

1 \pseudocodeblock{
2   \textbf{ Alice} \< \< \textbf{ Bob}   \\\[0.1\baselineskip][\hline]
3   \<\< \\\[-0.5\baselineskip]
4   x \sample \ZZ_q \< \< \\\
5   X \gets g^x \<\< \\\
6   \< \sendmessageright*{\GG,q,g,X} \< \\\
7   \<\< y \sample \ZZ_q \\\
8   \<\< Y \gets g^y \\\
9   \< \sendmessageleft*{Y} \< \\\
10  \key_A \gets Y^x \<\< \key_B \gets X^y }

```

We will discuss protocols in greater detail in Section 5.

1.5 Game-Based Proofs

Cryptocode supports authors in visualizing game-based proofs. It defines an environment `gameproof` which allows to wrap a number of game procedures displaying helpful information as to what changes from game to game and to what each step is reduced.

$\text{Game}_1(n)$	$\text{Game}_2(n)$	$\text{Game}_3(n)$	$\text{Game}_4(n)$
1 : Step 1	Step 1		Step 1
2 :	From game 3 on		From game 3 on
3 : Step 2	Step 3 is different		Step 3 adapted again

```

1 \begin{gameproof}
2   \begin{pchstack}[center,space=1em]
3     \gameprocedure[linenumbering,minlineheight=1.5em]{%
4       \text{Step 1} \\\
5       \\\
6       \text{Step 3}
7     }
8
9     \tbxgameprocedure[minlineheight=1.5em]{%
10      \text{Step 1} \\\
11      \pcbox{\text{From game 3 on}} \\\
12      \gamechange{\text{Step 3 is different}}
13    }
14
15    \gameprocedure[minlineheight=1.5em]{%
16      \text{Step 1} \\\
17      \text{From game 3 on} \\\
18      \text{Step 3 adapted again}
19    }
20
21 \end{pchstack}
22 \end{gameproof}

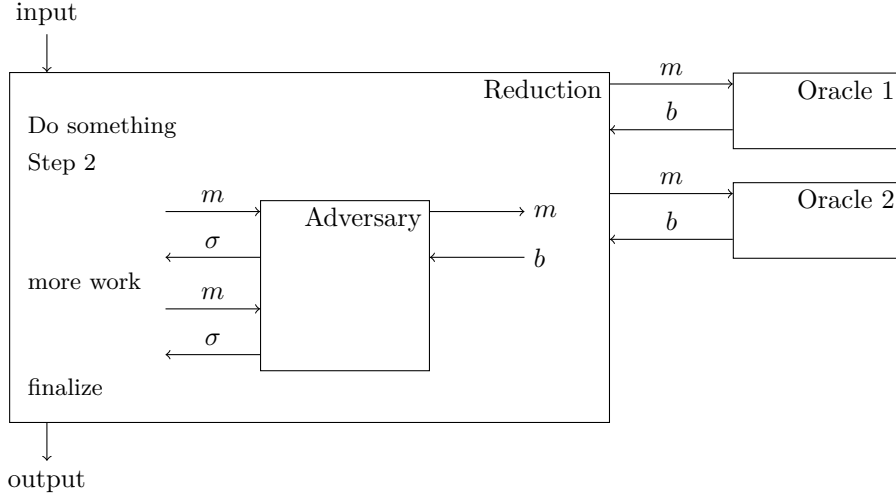
```

Note that we made use of the option “mode=text” in the above example which tells the underlying pseudocode command to not work in math mode but in plain text mode. We will discuss how to visualize game-based proofs in Section 6.

1.6 Black-Box Reductions

Cryptocode provides a structured syntax to visualize black-box reductions. Basically cryptocode provides an environment to draw boxes that may have oracles and/or challengers and that can be communicated with. Cryptocode makes heavy use of TIKZ (<https://www.ctan.org/pkg/pgf>) for this, which gives you quite some control over how things should look like. Additionally, as you can specify node names (for example the outer box in the next example is called “A”) you can easily extend the pictures by using

plain TIKZ commands. Following is an example reduction. We discuss the details in Section 7.



```

1 \begin{bbrenv}{A}
2   \begin{bbrbox}[name=Reduction]
3     \pseudocode{
4       \text{Do something} \\
5       \text{Step 2}
6     }
7
8     \begin{bbrenv}{B}
9       \begin{bbrbox}[name=Adversary ,minheight=2.25cm]
10        \end{bbrbox}
11
12        \bbrmsgto{top=$m$}
13        \bbrmsgfrom{top=$\sigma$}
14        \bbrmsgtxt{\pseudocode{\text{more work}}}
15      }
16    }
17    \bbrmsgto{top=$m$}
18    \bbrmsgfrom{top=$\sigma$}
19
20    \bbrqryto{side=$m$}
21    \bbrqryfrom{side=$b$}
22  \end{bbrenv}
23
24  \pseudocode{
25    \text{finalize}
26  }
27
28  \end{bbrbox}
29  \bbrinput{input}
30  \bbroutput{output}
31
32  \begin{bbroracle}{OraA}
33    \begin{bbrbox}[name=Oracle 1,minheight=1cm]
34    \end{bbrbox}
35  \end{bbroracle}
36  \bbroracleqryto{top=$m$}
37  \bbroracleqryfrom{top=$b$}
38
39  \begin{bbroracle}{OraB}
40    \begin{bbrbox}[name=Oracle 2,minheight=1cm]
41    \end{bbrbox}
42  \end{bbroracle}
43  \bbroracleqryto{top=$m$}
44  \bbroracleqryfrom{top=$b$}
45 \end{bbrenv}

```

2 Notation Macros

In this section we'll discuss the various commands for notation that can be loaded via package options.

```

1 \usepackage[
2   n, % or lambda
3   advantage,
4   operators,
5   sets,
6   adversary,
7   landau,
8   probability,
9   notions,
10  logic,
11  ff,
12  mm,
13  primitives,
14  events,
15  complexity,
16  oracles,
17  asymptotics,
18  keys
19 ]{cryptocode}

```

Remark. Note that the available command sets are far from complete and reflect my own work (especially once you get to cryptographic notions and primitives). In case you feel that something should be added feel free to drop me an email, or better yet, open an issue and pull request on github (<https://github.com/arnomi/cryptocode>).

2.1 Security Parameter

In cryptography we make use of a security parameter which is usually denoted by 1^n or 1^λ . The cryptocode package, when loading either option “n” or option “lambda” will define the commands

```

1 \secpa
2 \secpa
3 \SECPAR

```

The first command provides the “letter”, i.e., either n or λ , whereas `\secpa` prints 1^n (i.e., 1^n for option “n”). Finally, `\SECPAR` yields N_0 (resp. Λ) and is meant to be used in sentences such as, “there exists $N_0 \in \mathbb{N}$ such that for all $n \geq N_0$, ...”

2.2 Advantage Terms

Load the package option “advantage” in order to define the command `\advantage` used to specify advantage terms such as:

$$\text{Adv}_{\mathcal{A}, \text{PRF}}^{\text{prf}}(n)$$

```

1 \advantage{prf}{\adv,\prf}

```

Specify an optional third parameter to replace the (n) .

```

1 \advantage{prf}{\adv,\prf}[(arg)]

```

In order to redefine the styles in which superscript and subscript are set, or in case you want to replace the term `Adv`, redefine:

```

1 \renewcommand{\pcadvantagename}{\mathsf{Adv}}
2 \renewcommand{\pcadvantagesuperstyle}[1]{\mathrm{\MakeLowercase{#1}}}
3 \renewcommand{\pcadvantagesubstyle}[1]{#1}

```

2.3 Math Operators

The “operators” option provides the following list of commands:

Command	Description	Result	Example
<code>\sample</code>	Sampling from a distribution, or running a randomized procedure	$\leftarrow \$$	$b \leftarrow \$ \{0, 1\}$
<code>\floor{42.5}</code>	Rounding down	$\lfloor 42.5 \rfloor$	
<code>\ceil{41.5}</code>	Rounding up	$\lceil 41.5 \rceil$	
<code>\Angle{x,y}</code>	Vector product	$\langle x, y \rangle$	
<code>\abs{\frac{a}{b}}</code>	Absolute number	$\left \frac{a}{b} \right $	
<code>\norm{x}</code>	Norm	$\ x\ $	
<code>\concat</code>	Verbose concatenation (I usually prefer simply <code>\ </code>)	$\ $	$x \leftarrow a \ b$
<code>\emptystring</code>	The empty string	ε	$x \leftarrow \varepsilon$
<code>\argmax</code>	arg max	arg max	$\arg \max_{x \in S} f(x)$
<code>\argmin</code>	arg min	arg min	$\arg \min_{x \in S} f(x)$
<code>\pindist</code>	Perfect indistinguishability	$\underline{\approx}$	$X \underline{\approx} Y$
<code>\sindist</code>	Statistical indistinguishability	$\overset{s}{\approx}$	$X \overset{s}{\approx} Y$
<code>\cindist</code>	Computational indistinguishability	$\overset{c}{\approx}$	$X \overset{c}{\approx} Y$

The paired operators `\floor`, `\ceil`, `\Angle`, `\norm`, and `\abs` also come in a form for flow text which does not scale the outer delimiter. These are `\tfloor`, `\tceil`, `\tAngle`, `\tnorm`, and `\tabs`.

Note that `arg max` and `arg min` in block formulas will set their subscripts as limits, i.e.,:

$$\arg \max_{x \in S} f(x)$$

2.4 Adversaries

The “adversary” option provides the following list of commands:

Command	Description	Result
<code>\adv</code>	Adversary	\mathcal{A}
<code>\bdv</code>	Adversary	\mathcal{B}
<code>\cdv</code>	Adversary	\mathcal{C}
<code>\ddv</code>	Adversary	\mathcal{D}
<code>\edv</code>	Adversary	\mathcal{E}
<code>\mdv</code>	Adversary	\mathcal{M}
<code>\pdv</code>	Adversary	\mathcal{P}
<code>\rdv</code>	Adversary	\mathcal{R}
<code>\sdv</code>	Adversary	\mathcal{S}

The style in which an adversary is rendered is controlled via

```

1 \renewcommand{\pcadvstyle}[1]{\ensuremath{\mathcal{#1}}}

```

2.5 Landau

The “landau” option provides the following list of commands:

Command	Description	Result
<code>\bigO{n^2}</code>	Big O(micron) notation	$\mathcal{O}(n^2)$
<code>\smallO{n^2}</code>	small o(micron) notation	$\mathfrak{o}(n^2)$
<code>\bigOmega{n^2}</code>	Big Omega notation	$\Omega(n^2)$
<code>\bigTheta{n^2}</code>	Big Theta	$\Theta(n^2)$
<code>\orderOf</code>	On the order of	$f(n) \sim g(n)$

2.6 Probabilities

The “probability” option provides commands for writing probabilities. Use

```
1 \prob{X=x}
2 \probsub{x\sample{\bin^n}}{x=5}
3 \condprob{X=x}{A=b}
4 \condprobsub{x\sample{\bin^n}}{x=5}{A=b}
```

to write basic probabilities, probabilities with explicit probability spaces and conditional probabilities.

$$\begin{aligned} \Pr[X = x] \\ \Pr_{x \leftarrow \{0,1\}^n}[X = x] \\ \Pr[X = x \mid A = b] \\ \Pr_{x \leftarrow \{0,1\}^n}[x = 5 \mid A = b] \end{aligned}$$

You can control the probability symbol (\Pr) by redefining

```
1 \renewcommand{\probnname}{\Pr}
```

The probability commands have a flowtext version `\tprob{X=X}` or `\tcondprob{X=x}{Y=y}` which does not scale the delimiters. In case the probability space is more complex, you can use

```
1 \probsublong{x,y\sample\set{1,2,3,4,5,6}, z = x + y}{z = 7}
```

which yields

$$\Pr[z = 7 : x, y \leftarrow \{1, 2, 3, 4, 5, 6\}, z = x + y].$$

For specifying expectations the following commands are defined

```
1 \expect{X}
2 \expsub{x,y\sample\set{1,\ldots,6}}{x+y}
3 \condexp{X+Y}{Y>3}
4 \condexpsub{x,y\sample\set{1,\ldots,6}}{x+y}{y>3}
```

yielding

$$\begin{aligned} \mathbb{E}[X] \\ \mathbb{E}_{x,y \leftarrow \{1,\dots,6\}}[x + y] \\ \mathbb{E}[X + Y \mid Y > 3] \\ \mathbb{E}_{x,y \leftarrow \{1,\dots,6\}}[x + y \mid y > 3] \end{aligned}$$

Again flowtext versions such as `\texpect{X}` are available. To control the expectation symbol (\mathbb{E}), redefine

```
1 \renewcommand{\expectationname}{\ensuremath{\mathbb{E}}}
```

The support $\text{Supp}(X)$ of a random variable X can be written as

```
1 \supp{X}
```

where again the name can be controlled via

```
1 \renewcommand{\supportname}{Supp}
```

For denoting entropy and min-entropy use

```
1 \entropy{X}
2 \minentropy{X}
3 \condentropy{X}{Y=5}
4 \condminentropy{X}{Y=5}
5 \condavgminentropy{X}{Y=5}
```

This yields

$$\begin{aligned} H(X) \\ H_\infty(X) \\ H(X \mid Y = 5) \\ H_\infty(X \mid Y = 5) \\ \tilde{H}_\infty(X \mid Y = 5) \end{aligned}$$

2.7 Sets

The “sets” option provides commands for basic mathematical sets. You can write sets and sequences as

```
1 \set{1, \ldots, 10}
2 \sequence{1, \ldots, 10}
```

which are typeset as

$$\begin{aligned} \{1, \dots, 10\} \\ (1, \dots, 10) \end{aligned}$$

In addition, the following commands are provided

Command	Description	Result
<code>\bin</code>	The set containing 0 and 1	$\{0, 1\}$
<code>\NN</code>	Natural numbers	\mathbb{N}
<code>\ZZ</code>	Integers	\mathbb{Z}
<code>\QQ</code>	Rational numbers	\mathbb{Q}
<code>\CC</code>	Complex numbers	\mathbb{C}
<code>\RR</code>	Reals	\mathbb{R}
<code>\PP</code>		\mathbb{P}
<code>\FF</code>		\mathbb{F}
<code>\GG</code>		\mathbb{G}

The style in which sets are being set can be adapted by redefining

```
1 \renewcommand{\pcsetstyle}[1]{\ensuremath{\mathbb{#1}}}
```

2.8 Cryptographic Notions

The “notions” option defines the following list of commands:

Command	Description	Result
<code>\indcpa</code>	IND-CPA security for encryption schemes	IND-CPA
<code>\indcca</code>	IND-CCA security for encryption schemes	IND-CCA
<code>\indccai</code>	IND-CCA1 security for encryption schemes	IND-CCA1
<code>\indccaii</code>	IND-CCA2 security for encryption schemes	IND-CCA2
<code>\ind</code>	IND security	IND
<code>\priv</code>	PRIV security for deterministic public-key encryption schemes	PRIV
<code>\prvcda</code>	PRV-CDA security (for deterministic public-key encryption schemes)	PRV-CDA
<code>\prvrda</code>	PRV-CDA security (for deterministic public-key encryption schemes)	PRV-CDA
<code>\kia</code>	Key independent authenticated encryption	KIAE
<code>\kda</code>	Key dependent authenticated encryption	KDAE
<code>\mle</code>	Message locked encryption	MLE
<code>\uce</code>	Universal computational extractors	UCE
<code>\eufcma</code>	Existential unforgeability under chosen message attack	EUFCMA
<code>\eufnacma</code>	Non-adaptive existential unforgeability under chosen message attack	EUFCMA
<code>\seufcma</code>	Strong existential unforgeability under chosen message attack	SUFCMA
<code>\eufko</code>	Existential unforgeability under key only attack	EUFCMA

The style in which notions are displayed can be controlled via redefining

```
1 \renewcommand{\pcnotionstyle}[1]{\ensuremath{\mathrm{#1}}}
```

2.9 Logic

The “logic” option provides the following list of commands:

Command	Description	Result
<code>\AND</code>	Logical AND	AND
<code>\NAND</code>	Logical NAND	NAND
<code>\OR</code>	Logical OR	OR
<code>\NOR</code>	Logical NOR	NOR
<code>\XOR</code>	Logical XOR	XOR
<code>\XNOR</code>	Logical XNOR	XNOR
<code>\notimplies</code>	Negated implication	$\not\Rightarrow$
<code>\NOT</code>	not	NOT
<code>\xor</code>	exclusive or	\oplus
<code>\false</code>	false	false
<code>\true</code>	true	true

2.10 Function Families

The “ff” option provides the following list of commands:

Command	Description	Result
<code>\kgen</code>	Key generation	KGen
<code>\pgen</code>	Parameter generation	Pgen
<code>\eval</code>	Evaluation	Eval
<code>\invert</code>	Inversion	Inv
<code>\il</code>	Input length	il
<code>\ol</code>	Output length	ol
<code>\kl</code>	Key length	kl
<code>\nl</code>	Nonce length	nl
<code>\rl</code>	Randomness length	rl

The style in which these are displayed can be controlled via redefining

```
1 \renewcommand{\pcalgostyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.11 Machine Model

The “mm” option provides the following list of commands:

Command	Description	Result
<code>\CRKT</code>	A circuit	C
<code>\TM</code>	A Turing machine	M
<code>\PROG</code>	A program	P
<code>\uTM</code>	A universal Turing machine	UM
<code>\uC</code>	A universal Circuit	UC
<code>\uP</code>	A universal Program	UEval
<code>\tmtime</code>	Time (of a TM)	time
<code>\ppt</code>	Probabilistic polynomial time	PPT

The style in which these are displayed can be controlled via redefining

```
1 \renewcommand{\pcmachinestyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.12 Crypto Primitives

The “primitives” option provides the following list of commands:

Command	Description	Result
<code>\prover</code>	Proover	P
<code>\verifier</code>	Verifier	V
<code>\nizk</code>	Non interactive zero knowledge	NIZK
<code>\hash</code>	A hash function	H
<code>\gash</code>	A hash function	G
<code>\fash</code>	A hash function	F
<code>\pad</code>	A padding function	pad
<code>\enc</code>	Encryption	Enc
<code>\dec</code>	Decryption	Dec
<code>\sig</code>	Signing	Sig
<code>\sign</code>	Signing	Sign
<code>\verify</code>	Verifying	Vf
<code>\owf</code>	One-way function	OWF
<code>\prf</code>	Pseudorandom function	PRF
<code>\prp</code>	Pseudorandom permutation	PRP
<code>\prg</code>	Pseudorandom generator	PRG
<code>\obf</code>	Obfuscation	O
<code>\iO</code>	Indistinguishability obfuscation	iO
<code>\diO</code>	Differing inputs obfuscation	diO
<code>\mac</code>	Message authentication	MAC
<code>\puncture</code>	Puncturing	Puncture
<code>\source</code>	A source	S
<code>\predictor</code>	A predictor	P
<code>\sam</code>	A sampler	Sam
<code>\distinguisher</code>	A distinguisher	Dist
<code>\dist</code>	A distinguisher	D
<code>\simulator</code>	A simulator	Sim
<code>\extractor</code>	An extractor	Ext
<code>\ext</code>	Shorthand for <code>\extractor</code>	Ext

The style in which these are displayed can be controlled via redefining

```
1 \renewcommand{\pcalgostyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.13 Oracles

The “oracles” option provides the following list of commands:

Command	Description	Result
<code>\oracle</code>	Generic oracle	O
<code>\oracle[LoR]</code>	Custom oracle	LoR
<code>\ro</code>	Random oracle	RO
<code>\Oracle{\sign}</code>	Oracle version of procedure	O _{Sign}

The style in which these are displayed can be controlled via redefining

```
1 \renewcommand{\pcoraclestyle}[1]{\ensuremath{\mathsf{#1}}}
```

2.14 Events

The “events” option provides the following list of commands.

Command	Description	Result
<code>\event{E}</code>	Event E	E
<code>\nevent{E}</code>	Negated event E	\bar{E}
<code>\bad</code>	Bad event	bad
<code>\nbad</code>	Bad event	$\overline{\text{bad}}$

2.15 Complexity

The “complexity” option provides the following list of commands:

Command	Result
<code>\complclass{myClass}</code>	myClass
<code>\cocomplclass{myClass}</code>	co-myClass
<code>\npol</code>	NP
<code>\conpol</code>	co-NP
<code>\pol</code>	P
<code>\bpp</code>	BPP
<code>\ppoly</code>	P/poly
<code>\NC{1}</code>	NC^1
<code>\AC{1}</code>	AC^1
<code>\TC{1}</code>	TC^1
<code>\AM</code>	AM
<code>\coAM</code>	co-AM
<code>\PH</code>	PH
<code>\csigma{1}</code>	Σ_1^p
<code>\cpi{1}</code>	Π_1^p
<code>\cosigma{1}</code>	$\text{co-}\Sigma_1^p$
<code>\copi{1}</code>	$\text{co-}\Pi_1^p$

The style in which these are displayed can be controlled via redefining

¹ `\renewcommand{\pccomplexitystyle}[1]{\ensuremath{\mathsf{\#1}}}`

2.16 Asymptotics

The “asymptotics” option provides the following list of commands:

Command	Description	Result
<code>\negl</code>	A negligible function	$\text{negl}(n)$ (n is <code>\secpar</code>)
<code>\negl[x]</code>	A negligible function	$\text{negl}(x)$
<code>\negl[]</code>	A negligible function	negl
<code>\poly</code>	A polynomial	$\text{poly}(n)$ (n is <code>\secpar</code>)
<code>\poly[x]</code>	A polynomial	$\text{poly}(x)$
<code>\poly[]</code>	A polynomial	poly
<code>\pp</code>	some polynomial p	p
<code>\pp[t]</code>	some custom polynomial t	t
<code>\cc</code>	some polynomial c	c
<code>\ee</code>	some polynomial e	e
<code>\kk</code>	some polynomial k	k
<code>\mm</code>	some polynomial m	m
<code>\nn</code>	some polynomial n	n
<code>\qq</code>	some polynomial q	q
<code>\rr</code>	some polynomial r	r

The style in which these are displayed can be controlled via redefining

```
1 \renewcommand{\pcpolynomialstyle}[1]{\ensuremath{\mathrm{\mathstrut{#1}}}}
```

2.17 Keys

The “keys” option provides the following list of commands:

Command	Description	Result
<code>pk</code>	public key	pk
<code>vk</code>	verification key	vk
<code>sk</code>	secret key	sk
<code>key</code>	a plain key	k
<code>key[xk]</code>	custom key	xk
<code>hk</code>	hash key	hk
<code>gk</code>	gash key	gk
<code>fk</code>	function key	fk
<code>st</code>	state	st
<code>state</code>	state	$state$
<code>state{myState}</code>	custom state	$state_{myState}$

The style in which these are displayed can be controlled via redefining

```
1 \renewcommand{\pckeystyle}[1]{\ensuremath{\mathsf{\mathstrut{#1}}}}
```

3 Pseudocode

3.1 Basics

The cryptocode package provides the command `\pseudocode` for typesetting algorithms. Consider the following definition of an IND-CPA game

$$\begin{aligned}
 b &\leftarrow \$ \{0, 1\} \\
 (\mathbf{pk}, \mathbf{sk}) &\leftarrow \$ \text{KGen}(1^n) \\
 (m_0, m_1) &\leftarrow \$ \mathcal{A}(1^n, \mathbf{pk}, c) \\
 c &\leftarrow \$ \text{Enc}(\mathbf{pk}, m_b) \\
 b' &\leftarrow \$ \mathcal{A}(1^n, \mathbf{pk}, c) \\
 \textbf{return } b &= b'
 \end{aligned}$$

which is generated by

```

1 \begin{pchstack}[center]
2 \pseudocode{
3   b \sample \bin \\
4   (\pk,\sk) \sample \kgen (\seccparam) \\
5   (m_0,m_1) \sample \adv(\seccparam, \pk, c) \\
6   c \sample \enc(\pk,m_b) \\
7   b' \sample \adv(\seccparam, \pk, c) \\
8   \pcreturn b = b' }
9 \end{pchstack}

```

First note that `\pseudocode` on its own does not space itself. For laying out one (or multiple) code blocks cryptocode defines stacking environments such as `\pchstack` and `\pcvstack` that we discuss in Section 3.7. Wrapping a single pseudocode in a `\pchstack` as in the above example generates a nicely offset code block.

As code blocks are most often not used in flow text, cryptocode offers the shorthand `\pseudocodeblock` which centers and offsets a pseudocode block as above. We thus get the very same by writing

```

1 \pseudocodeblock{
2   b \sample \bin \\
3   (\pk,\sk) \sample \kgen (\seccparam) \\
4   (m_0,m_1) \sample \adv(\seccparam, \pk, c) \\
5   c \sample \enc(\pk,m_b) \\
6   b' \sample \adv(\seccparam, \pk, c) \\
7   \pcreturn b = b' }

```

We can also define custom block commands, for example, the following defines a command `\pcb` that offsets and centers code and draws a tight fitting box around the code block:

```

1 \createpseudocodeblock{pcb}{center,boxed}{\}\{\}\{\}

```

(We discuss creating custom pseudocode commands in detail in Section 3.1.2). If we now use `\pcb` as just defined in the above example, we obtain the following nicely spaced and boxed result.

```

 $b \leftarrow \{0, 1\}$ 
 $(pk, sk) \leftarrow \text{KGen}(1^n)$ 
 $(m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$ 
 $c \leftarrow \text{Enc}(pk, m_b)$ 
 $b' \leftarrow \mathcal{A}(1^n, pk, c)$ 
return  $b = b'$ 

```

which is generated as

```

1 \pcb{
2   b \sample \bin \\
3   (\pk,\sk) \sample \kgen (\secpam) \\
4   (m_0,m_1) \sample \adv(\secpam, \pk, c) \\
5   c \sample \enc(\pk,m_b) \\
6   b' \sample \adv(\secpam, \pk, c) \\
7   \pcreturn b = b' }

```

Remark. In the following we will use this boxed representation for the examples, but use `\pseudocodeblock` in the corresponding code listings.

As you can see, the pseudocode command provides a math based environment where you can simply start typing your pseudocode separating lines by `\\`.

3.1.1 Customizing Pseudocode

Besides the mandatory argument the `\pseudocode` command can take an optional argument which consists of a list of key=value pairs separated by commas.

```

1 \pseudocode[options]{body}

```

The following parameters are available:

head A header for the code

width An exact width. If no width is specified, cryptocode tries to automatically compute the correct width.

lstart The starting line number when using line numbering.

lstarttright The starting line number for right aligned line numbers when using line numbering.

linenumbering Enables line numbering.

skipfirstln Starts line numbering on the second line.

minlineheight Specify a minimum height for each line. Can be globally set by redefining `\pcminlineheight`.

syntaxhighlight When set to “auto” cryptocode will attempt to automatically highlight keywords such as “for”, “foreach” and “return”. Note that this feature should be regarded as experimental. In particular, it is rather slow.

keywords Provide a comma separated list of keywords for automatic syntax highlighting. To customize the behavior of automatic spacing you can provide keywords as

keywordsindent After seeing this keyword all following lines will be indented one extra level.

keywordsunindent After seeing this keyword the current and all following lines will be unindented one extra level.

keywordsuninindent After seeing this keyword the current line will be unindented one level.

addkeywords Provide additional keywords for automatic syntax highlighting.

altkeywords Provide a second list of keywords for automatic syntax highlighting that are highlighted differently.

mode When set to text pseudocode will not start in math mode but in text mode.

space Allows you to enable automatic spacing mode. If set to “keep” the spaces in the input are preserved. If set to “auto” it will try to detect spacing according to keywords such as “if” and “fi”.

codesize Allows to specify the fontsize for the pseudocode. Set to `\scriptsize` for a smaller size.

colspace Allows to insert spacing between columns. In particular this allows to also overlap columns by inserting negative space.

jot Allows to specify extra space between each line. Use `jot=1mm`.

beginline Allows to specify a macro that is placed at the beginning of each line.

endline Allows to specify a macro that is placed at the end of each line.

xshift Allows horizontal shifting

yshift Allows horizontal shifting

headlinesep Specifies the distance between header and the line. By default set to 0pt which can be globally overwritten by setting length `\pcheadlinesep`.

bodylinesep Specifies the distance between body and the line. By default set to `0.3\baselineskip` which can be globally overwritten by setting length `\pcbodylinesep`.

colsep Defines the space between columns.

headheight Specifies the height of the header. By default set to 3.25ex which can be globally overwritten by setting length `\pcheadheight`.

headlinecmd Allows to overwrite which command is used to draw the bar below the headline. Defaults to `\hrule`.

addtolength Is added to the automatically computed width of the pseudocode (which does not take colsep into account).

valign Controls the vertical alignment of the pseudocode. Pseudocode is wrapped in a minipage environment and valign value is passed as orientation for the minipage. By default valign is set to “t”.

nodraft Forces syntax highlighting also in draft mode.

The following code

```
1 \pseudocodeblock[linenumbering,syntaxhighlight=auto,head=Header]{ return null }
```

creates

Header
1 : return null

3.1.2 Customized Pseudocode Commands

Besides the `\pseudocode` and `\pseudocodeblock` command the command `\procedure` (and its block variant `\procedureblock` provides easy access to generate code with a header. They take the following form

```
1 \procedure[options]{Header}{Body}
2 \procedureblock[options]{Header}{Body}
```

Examples

IND-CPA _{Enc} ^A (n)
$b \leftarrow \{0, 1\}$ $(pk, sk) \leftarrow \text{KGen}(1^n)$ $(m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$ $c \leftarrow \text{Enc}(pk, m_b)$ $b' \leftarrow \mathcal{A}(1^n, pk, c)$ return $b = b'$

which is generated as

```
1 \procedureblock{$\indcpa\_enc^{\sim}\adv(\secpa r)$}{
2   b \sample \bin \
3   (\pk,\sk) \sample \kgen(\secpa r) \
4   (m_0,m_1) \sample \adv(\secpa r, \pk, c) \
5   c \sample \enc(\pk,m_b) \
6   b' \sample \adv(\secpa r, \pk, c) \
7   \pcreturn b = b' }
```

You can define customized pseudocode commands which either take one optional argument and two mandatory arguments (as the procedure command) or one optional and one mandatory argument (as the pseudocode command). The following

```
1 \createpseudocodecommand{mypseudocode}{}{}{linenumbering}
2 \createprocedurecommand{myprocedure}{}{}{linenumbering}
3 \createpseudocodeblock{pcb}{center,boxed}{}{}{linenumbering}
4 \createprocedureblock{procb}{center,boxed}{}{}{linenumbering}
```

creates the commands `\mypseudocode` and `\myprocedure` with line numbering always enabled as well as the block commands `\pcb` and `\procb` also with line numbering enabled. The created commands have an identical interface as the `\pseudocode` (resp. `\procedure`) command. The two arguments that we kept empty when generating the commands allows us to specify commands that are executed at the very beginning when the command is called (first empty argument) and a prefix for the header. For example, the command created as


```
1 \createprocedureblock{expproc}{center,boxed}{\mathrm{Experiment}\xspace}{
  \linenumbers}
```

could be used as

```
1 \expproc{\indcpa_\enc^{\adv(\secpa)}}{
2   b \sample \bin \\
3   (\pk,\sk) \sample \kgen(\secpa) \\
4   (m_0,m_1) \sample \adv(\secpa,\pk,c) \\
5   c \sample \enc(\pk,m_b) \\
6   b' \sample \adv(\secpa,\pk,c) \\
7   \pcreturn b = b' }
```

This results in

Experiment IND-CPA _{Enc} ^A (n)
1 : $b \leftarrow \{0, 1\}$
2 : $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$
3 : $(m_0, m_1) \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
4 : $c \leftarrow \text{Enc}(\text{pk}, m_b)$
5 : $b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
6 : return $b = b'$

3.2 Indentation

In order to indent code use `\pcind` or short `\t`. You can also use customized spacing such as `\quad` or `\hspace` when using the pseudocode command in math mode.

for $i = 1..10$ do $T[i] \leftarrow \{0, 1\}^n$ for $i = 1..10$ do $T[i] \leftarrow \{0, 1\}^n$
--

which is generated as

```
1 \pseudocodeblock{
2   \pcfor i = 1..10 \pcdo \\
3   \pcind T[i] \sample \bin^n \\
4   \pcfor i = 1..10 \pcdo \\
5   \t T[i] \sample \bin^n }
```

You can specify multiple levels via the optional first argument

```
1 \t[level] % \pcind[level]
```

for $i = 1..10$ do $T[i] \leftarrow \{0, 1\}^n$ $T[i] \leftarrow \{0, 1\}^n$ $T[i] \leftarrow \{0, 1\}^n$ $T[i] \leftarrow \{0, 1\}^n$ $T[i] \leftarrow \{0, 1\}^n$
--

```

1 \pseudocodeblock{
2   \pfor i = 1..10 \pdo  \\
3   \t T[i] \sample \bin^n \\
4   \t\t T[i] \sample \bin^n \\
5   \t[3] T[i] \sample \bin^n \\
6   \t[4] T[i] \sample \bin^n \\
7   \t[5] T[i] \sample \bin^n }

```

You can customize the indentation shortcut by redefining

```

1 \renewcommand{\pcindentname}{t}

```

Automatic Indentation

The pseudocode command comes with an option “space=auto” which tries to detect the correct indentation from the use of keywords. When it sees one of the following keywords

```

1 \pcif , \pfor , \pcwhile , \pcrepeat , \pcforeach

```

it will increase the indentation starting from the next line. It will again remove the indentation on seeing

```

1 \pcfi , \pcendif , \pcendfor , \pcendwhile , \pcuntil , \pcendforeach

```

Additionally, on seeing

```

1 \pcelse , \pcelseif

```

it will remove the indentation for that particular line. Thus the following

```

for  $a \in [10]$  do
  for  $a \in [10]$  do
    for  $a \in [10]$  do
      if  $a = b$  then
        some operation
      elseif  $a = c$  then
        some operation
      else
        some default operation
      fi
    endfor
  endfor
endfor
return  $a$ 

```

can be obtained by:

```

1 \pseudocodeblock[space=auto]{%
2 \pfor a \in [10] \pdo \\
3 \pfor a \in [10] \pdo \\
4 \pfor a \in [10] \pdo \\
5 \pcif a = b \pcthen \\
6 \text{some operation} \\

```

```

7      \pcelseif a = c \pcthen \\
8      \text{some operation} \\
9      \pcelse \\
10     \text{some default operation} \\
11     \pcfi \\
12     \pcendfor \\
13     \pcendfor \\
14     \pcendfor \\
15     \pcreturn a}

```

Note that the manual indentation in the above example is not necessary for the outcome. Further note that the same works when using automatic syntax highlighting (see Section 3.4).

Keep Input Indentation (experimental)

The pseudocode package comes with an *experimental* feature that preserves the spacing in the input. This can be enabled with the option “space=keep”.

```

1 \begin{center}
2 \pseudocode[space=keep]{%
3 \pcfor i = 1..10 \pcdo \\
4   T[i] \sample \bin^n \\
5     T[i] \sample \bin^n \\
6       T[i] \sample \bin^n \\
7         T[i] \sample \bin^n \\
8           T[i] \sample \bin^n }
9 \end{center}

```

This yields the following result

```

for i = 1..10 do
  T[i]  $\leftarrow \{0,1\}^n$ 
  T[i]  $\leftarrow \{0,1\}^n$ 
  T[i]  $\leftarrow \{0,1\}^n$ 
  T[i]  $\leftarrow \{0,1\}^n$ 
  T[i]  $\leftarrow \{0,1\}^n$ 

```

Note that automatic spacing only works when the `\pseudocode` command is not wrapped within another command. Thus in order to get a frame box `\fbox{\pseudocode[space=keep]{code}}` will not work but you would need to use an environment such as one offered by the *mdframed* package (<https://www.ctan.org/pkg/mdframed>). Also see Section 8.1.

3.3 Textmode

By default pseudocode enables L^AT_EX’ math mode. You can change this behavior and tell the pseudocode command to interpret the content in text mode by setting the option “mode=text”.

This is
simply text

```

1 \pseudocodeblock[mode=text]{%
2 This is \\
3 \t simply text}

```

3.4 Syntax Highlighting

In the above examples we have used commands `\pcreturn` and `\pcfor` to highlight certain keywords. Besides the *pcreturn*, *pcfor* and *pcdo* (where the pc stands for pseudocode) that were used in the above examples the package defines the following set of constants:

command	outcome
<code>\pcabort</code>	abort
<code>\pcassert</code>	assert
<code>\pccontinue</code>	continue
<code>\pccomment{comment}</code>	// comment
<code>\pccomment[2em]{comment}</code>	// comment
<code>\pclinecomment{comment}</code>	// comment
<code>\pcdo</code>	do
<code>\pcdone</code>	done
<code>\pcfail</code>	fail
<code>\pcfalse</code>	false
<code>\pcif</code>	if
<code>\pcfi</code>	fi
<code>\pcendif</code>	endif
<code>\pcelse</code>	else
<code>\pcelseif</code>	elseif
<code>\pcfor</code>	for
<code>\pcendfor</code>	endfor
<code>\pcforeach</code>	foreach
<code>\pcendforeach</code>	endforeach
<code>\pcglobvar</code>	gbl
<code>\pcin</code>	in
<code>\pcnew</code>	new
<code>\pcnull</code>	null
<code>\pcparse</code>	parse
<code>\pcrepeat{10}</code>	repeat 10 times
<code>\pcreturn</code>	return
<code>\pcuntil</code>	until
<code>\pcthen</code>	then
<code>\pctrue</code>	true
<code>\pcwhile</code>	while
<code>\pcendwhile</code>	endwhile

Note that `\pcdo`, `\pcin` and `\pcthen` have a leading space. This is due to their usual usage scenarios such as

for *i* **in**{1,...,10}

Furthermore all constants have a trailing space. This can be removed by adding the optional parameter `[]` such as

for *i***in**{1,...,10}

```
1 \pseudocodeblock{\pcfor i \pcin [] \{1,\ldots,10\}}
```

In order to change the font you can overwrite the command `\highlightkeyword` which is defined as

```
1 \newcommand{\highlightkeyword}[2][\ ]{\ensuremath{\mathbf{#2}}#1}
```

3.4.1 Automatic Syntax Highlighting (Experimental)

The pseudocode command comes with an experimental (and rather slow) feature to automatically highlight keywords. This can be activated via the option “syntaxhighlight=auto”. The preset list of keywords it looks for are

```
1 for,foreach,{return },return,{ do },{ in },new,if, null, true,{ until },{ to },
false,{ then},repeat,else if,elseif,while,else,done
```

Note that the keywords are matched with spaces and note the grouping for trailing spaces. That is, the “ do ” keyword won’t match within the string “don’t”. Via the option “keywords” you can provide a custom list of keywords. Thus the following bubblesort variant (taken from http://en.wikipedia.org/wiki/Bubble_sort)

```
Bubblesort(A : list of items)
n ← length(A)
repeat
  s ← false
  for i = 1 to n - 1 do
    // if this pair is out of order
    if A[i - 1] > A[i] then
      // swap them and remember something changed
      swap(A[i - 1], A[i])
      s ← true
  until ¬s
```

can be typeset as

```
1 \procedureblock[syntaxhighlight=auto]{Bubblesort(A : list of items)}{
2   n \gets \mathsf{length}(A) \\\
3   repeat \\\
4     \t s \gets false \\\
5     \t for i = 1 to n-1 do \\\
6       \t\t \pcomment{if this pair is out of order} \\\
7       \t\t if A[i-1] > A[i] then \\\
8         \t\t\t \pcomment{swap them and remember something changed} \\\
9         \t\t\t \mathsf{swap}( A[i-1], A[i] ) \\\
10        \t\t\t s \gets true \\\
11    until \neg s }
```

You can also define additional keywords using the “addkeywords” option. This would allow us to specify “length” and “swap” in the above example.

```

Bubblesort(A : list of items)
n ← length(A)
repeat
  s ← false
  for i = 1 to n - 1 do
    // if this pair is out of order
    if A[i - 1] > A[i] then
      // swap them and remember something changed
      swap(A[i - 1], A[i])
      s ← true
  until ¬s

```

can be typeset as

```

1 \procedureblock[syntaxhighlight=auto,addkeywords={swap,length}]{Bubblesort(A :
2   list of items)}{
3   n \gets \mathsf{length}(A) \\
4   repeat \\
5     \t s \gets false \\
6     \t for i = 1 to n-1 do \\
7       \t\t \pcomment{if this pair is out of order} \\
8       \t\t if A[i-1] > A[i] then \\
9         \t\t\t \pcomment{swap them and remember something changed} \\
10        \t\t\t \mathsf{swap}(A[i-1], A[i]) \\
11        \t\t\t s \gets true \\
12      until \neg s }

```

We can also combine automatic syntax highlighting with automatic spacing in which case we need to insert “group end” keywords:

```

Bubblesort(A : list of items)
n ← length(A)
repeat
  s ← false
  for i = 1 to n - 1 do
    // assuming this pair is out of order
    if A[i - 1] > A[i] then
      // swap them and remember something changed
      swap(A[i - 1], A[i])
      s ← true
    endif
  endfor
until ¬s

```

```

1 \procedureblock[space=auto,syntaxhighlight=auto,addkeywords={swap,length}]{
2   Bubblesort(A : list of items)}{
3   n \gets length(A) \\
4   repeat \\
5     s \gets false \\
6     for i=1 to n-1 do \\
7       \pcomment{assuming this pair is out of order} \\
8       if A[i-1]>A[i] then \\
9         \pcomment{swap them and remember something changed} \\

```

```

9         swap(A[i-1], A[i]) \\
10        s \gets true \\
11    endif \\
12  endfor \\
13 until \neg s }

```

Alternative Keywords

There is a second keyword list that you can add keywords to which are highlighted not via `\highlightkeyword` but via `\highlightaltkeyword` where alt stands for alternate. This allows you to have two different keyword styles which are by default defined as

```

1 \newcommand{\highlightkeyword}[2][\ ]{\ensuremath{\mathbf{#2}}#1}
2 \newcommand{\highlightaltkeyword}[1]{\ensuremath{\mathsf{#1}}}

```

This allows you to rewrite the above example and emphasize the different nature of swap and length.

```

Bubblesort(A : list of items)
n ← length (A)
repeat
  s ← false
  for i = 1 to n - 1 do
    // assuming this pair is out of order
    if A[i - 1] > A[i] then
      // swap them and remember something changed
      swap (A[i - 1], A[i])
      s ← true
    endif
  endfor
until ¬s

```

```

1 \procedureblock[space=auto,syntaxhighlight=auto,addkeywords={swap,length}]{
2   Bubblesort(A : list of items)}{
3   n \gets length(A) \\
4   repeat \\
5     s \gets false \\
6     for i=1 to n-1 do \\
7       \pcomment{assuming this pair is out of order} \\
8       if A[i-1]>A[i] then \\
9         \pcomment{swap them and remember something changed} \\
10        swap(A[i-1], A[i]) \\
11        s \gets true \\
12      endif \\
13    endfor \\
14  until \neg s }

```

Draft Mode

Automatic syntax highlighting is a somewhat expensive operation as it requires several rounds of regular expression matching. In order to speed up compilation the pseudocode command will not attempt automatic highlighting when the document is in draft mode. When in draft mode and you want to force a specific instance of `\pseudocode` to render the code with automatic syntax highlighting you can use the option `nodraft`.

3.5 Line Numbering

The pseudocode command allows to insert line numbers into pseudocode. You can either manually control line numbering or simply turn on the option `linenumbering`.

IND-CPA _{Enc} ^A (1 ⁿ)	
1 :	$b \leftarrow \$ \{0, 1\}$
2 :	$(pk, sk) \leftarrow \$ \text{KGen}(1^n)$
3 :	$(m_0, m_1) \leftarrow \$ \mathcal{A}(1^n, pk, c)$
4 :	$c \leftarrow \$ \text{Enc}(pk, m_b)$
5 :	$b' \leftarrow \$ \mathcal{A}(1^n, pk, c)$
6 :	return $b = b'$

is generated by

```

1 \procedureblock[linenumbering]{\indcpa_\enc^{\adv(\seccparam)}\{%
2   b \sample \bin \\\
3   (\pk,\sk) \sample \kgen(\seccparam) \\\
4   \label{my:line:label} (m_0,m_1) \sample \adv(\seccparam, \pk, c) \\\
5   c \sample \enc(\pk,m_b) \\\
6   b' \sample \adv(\seccparam, \pk, c) \\\
7   \pcreturn b = b' }
```

Note that you can use labels. In the above example `\label{my:line:label}` points to **3**.

3.5.1 Skipping Line Numbers

When using automatic line numbering, you can skip line numbers by inserting a `\pcskipln` command. This causes the line number on the *next line* to be suppressed. In order to suppress the first line number use the option `skipfirstln`. Thus the following

	// Some comment on first line
1 :	Some code
	// Some other comment
2 :	Some code

is generated by

```

1 \pseudocodeblock[linenumbering,skipfirstln,mode=text]{
2   \pclinecomment{Some comment on first line} \\\
3   Some code \pcskipln\\
4   \pclinecomment{Some other comment} \\\
5   Some code }
```

3.5.2 Manually Inserting Line Numbers

In order to manually insert line numbers use the command `\pcln`.

IND-CPA $_{\text{Enc}}^A(1^n)$	
1 :	$b \leftarrow \{0, 1\}$
2 :	$(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$
3 :	$(m_0, m_1) \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
4 :	$c \leftarrow \text{Enc}(\text{pk}, m_b)$
5 :	$b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
6 :	return $b = b'$

is generated by

```

1 \procedure{\$ \indcpa_\enc^adv(\secpam)}{
2 \pcln b \sample \bin \
3 \pcln (\pk, \sk) \sample \kgen(\secpam) \
4 \pcln \label{my:line:label2} (m_0, m_1) \sample \adv(\secpam, \pk, c) \
5 \pcln c \sample \enc(\pk, m_b) \
6 \pcln b' \sample \adv(\secpam, \pk, c) \
7 \pcln \pcreturn b = b' }
```

Note that labels also work when manually placing line numbers. In the above example label *my:line:label2* points to line number 3.

3.5.3 Start Values

You can specify the start value (minus one) of the counter by setting the option `lnstart`.

```

1 \procedure[lnstart=10,linenumbering]{Header}{Body}
```

IND-CPA $_{\text{Enc}}^A(1^n)$	
11 :	$b \leftarrow \{0, 1\}$
12 :	$(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$
13 :	$(m_0, m_1) \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
14 :	$c \leftarrow \text{Enc}(\text{pk}, m_b)$
15 :	$b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
16 :	return $b = b'$

3.5.4 Separators

The command `\pclnseparator` defines the separator between code and line number. By default the left separator is set to (:) colon. Also see Section 5.3.1.

3.5.5 Style

The style in which line numbers are set can be controlled by redefining `\pclnstyle`.

```

1 \renewcommand \pclnstyle [1]{\text{\scriptsize #1}}
```

For example, to set line numbers in normal font and dot separated use

```

1 \renewcommand{\pclnstyle}[1]{\text{#1}}
2 \renewcommand{\pclnseparator}{.}
```

IND-CPA _{Enc} ^A (1 ⁿ)
1. $b \leftarrow \$ \{0, 1\}$
2. $(pk, sk) \leftarrow \$ \text{KGen}(1^n)$
3. $(m_0, m_1) \leftarrow \$ \mathcal{A}(1^n, pk, c)$
4. $c \leftarrow \$ \text{Enc}(pk, m_b)$
5. $b' \leftarrow \$ \mathcal{A}(1^n, pk, c)$
6. return $b = b'$

3.6 Subprocedures

The pseudocode package allows the typesetting of subprocedures such as

IND-CPA _{Enc} ^A (1 ⁿ)
1 : $b \leftarrow \$ \{0, 1\}$
2 : $(pk, sk) \leftarrow \$ \text{KGen}(1^n)$
3 : $(m_0, m_1) \leftarrow \$ \mathcal{A}(1^n, pk, c)$
<div style="border: 1px dashed black; padding: 5px; margin-left: 100px;"> 1 : Step 1 2 : Step 2 3 : return m_0, m_1 </div>
4 : $c \leftarrow \$ \text{Enc}(pk, m_b)$
5 : $b' \leftarrow \$ \mathcal{A}(1^n, pk, c)$
6 : return $b = b'$

To create a subprocedure use the `subprocedure` environment. The above example is generated via

```

1 \procedureblock[linenumbering]{\indcpa_\enc^adv(\secpam)}{%
2   b \sample \bin \\\
3   (\pk,\sk) \sample \kgen(\secpam) \\\
4   (m_0,m_1) \sample \begin{subprocedure}%
5   \dbox{\procedure{\adv(\secpam, \pk, c)}{%
6     \text{Step 1} \\\
7     \text{Step 2} \\\
8     \pcreturn m_0, m_1 }}
9   \end{subprocedure} \\\
10  c \sample \enc(\pk,m_b) \\\
11  b' \sample \adv(\secpam, \pk, c) \\\
12  \pcreturn b = b' }
```

Here the `dbox` command (from the `dashbox` package) is used to generate a dashed box around the sub procedure.

3.6.1 Numbering in Subprocedures

As subprocedures are simply normal pseudocode blocks, you can use easily add line numbers. By default the line numbering starts with 1 in a subprocedure while ensuring that the outer numbering remains intact. Also note that the `linenumbering` on the outer procedure in the above example is inherited by the subprocedure. For more control, either use manual numbering or set the option “`linenumbering=off`” on the `\pseudocode` command within the subprocedure.

3.7 Stacking Procedures

You can stack procedures horizontally or vertically using the environments “pchstack” and “pcvstack”.

```
1 \begin{pchstack}[options] body \end{pchstack}
2 \begin{pcvstack}[options] body \end{pcvstack}
```

The following example displays two procedures next to one another. To space two horizontally outlined procedures use the `space` option or manually insert spaces via `\pchspace` which takes an optional length as a parameter.

IND-CPA $\mathcal{A}_{\text{Enc}}(1^n)$	Oracle O
1 : $b \leftarrow \{0, 1\}$	1 : Some code
2 : $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$	2 : Some more code
3 : $(m_0, m_1) \leftarrow \mathcal{A}^O(1^n, \text{pk})$	
4 : $c \leftarrow \text{Enc}(\text{pk}, m_b)$	
5 : $b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$	
6 : return $b = b'$	

```
1 \begin{pchstack}[boxed, center, space=1em]
2 \procedure[linenumbering]{\indcpa_\enc^adv(\secpam)}{
3   b \sample \bin \
4   (\pk, \sk) \sample \kgen(\secpam) \
5   (m_0, m_1) \sample \adv^O(\secpam, \pk) \
6   c \sample \enc(\pk, m_b) \
7   b' \sample \adv(\secpam, \pk, c) \
8   \pcreturn b = b' }
9
10 % alternatively use \pchspace for spacing
11
12 \procedure[linenumbering, mode=text]{Oracle \$O\$}{
13   Some code \
14   Some more code
15 }
16 \end{pchstack}
```

Similarly you can stack two procedures vertically using the “pcvstack” environment. As a spacing between two vertically stacked procedures again use either the `space` option or insert space manually via `\pcvspace` which takes an optional length as a parameter.

IND-CPA $\mathcal{A}_{\text{Enc}}(1^n)$
1 : $b \leftarrow \{0, 1\}$
2 : $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$
3 : $(m_0, m_1) \leftarrow \mathcal{A}^O(1^n, \text{pk})$
4 : $c \leftarrow \text{Enc}(\text{pk}, m_b)$
5 : $b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
6 : return $b = b'$
Oracle O
1 : Some code
2 : Some more code

```

1 \begin{pcvstack}[boxed,center,space=0.5em]
2 \procedure[linenumbering]{\indcpa_\enc^\adv(\secparam)}{%
3   b \sample \bin \\\
4   (\pk,\sk) \sample \kgen(\secparam) \\\
5   (m_0,m_1) \sample \adv^O(\secparam, \pk) \\\
6   c \sample \enc(\pk,m_b) \\\
7   b' \sample \adv(\secparam, \pk, c) \\\
8   \pcreturn b = b' }
9
10 % alternatively use \pcvspace for spacing
11
12 \procedure[linenumbering,mode=text]{Oracle \$O}{%
13   Some code \\\
14   Some more code
15 }
16 \end{pcvstack}

```

Horizontal and vertical stacking can be combined

$\text{IND-CPA}_{\text{Enc}}^A(1^n)$	$\text{IND-CPA}_{\text{Enc}}^A(1^n)$
1: $b \leftarrow \{0,1\}$	1: $b \leftarrow \{0,1\}$
2: $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$	2: $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^n)$
3: $(m_0, m_1) \leftarrow \mathcal{A}^O(1^n, \text{pk})$	3: $(m_0, m_1) \leftarrow \mathcal{A}^O(1^n, \text{pk})$
4: $c \leftarrow \text{Enc}(\text{pk}, m_b)$	4: $c \leftarrow \text{Enc}(\text{pk}, m_b)$
5: $b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$	5: $b' \leftarrow \mathcal{A}(1^n, \text{pk}, c)$
6: return $b = b'$	6: return $b = b'$

Oracle O	Oracle H_1	Oracle H_2
1: Some code	1: Some code	1: Some code
2: Some more code	2: Some more code	2: Some more code

```

1 \begin{pcvstack}[boxed,center,space=1em]
2 \begin{pchstack}[center,space=2em]
3
4   \procedure[linenumbering]{\indcpa_\enc^\adv(\secparam)}{%
5     b \sample \bin \\\
6     (\pk,\sk) \sample \kgen(\secparam) \\\
7     (m_0,m_1) \sample \adv^O(\secparam, \pk) \\\
8     c \sample \enc(\pk,m_b) \\\
9     b' \sample \adv(\secparam, \pk, c) \\\
10    \pcreturn b = b' }
11
12 % alternatively use \pchspace for spacing
13
14 \procedure[linenumbering]{\indcpa_\enc^\adv(\secparam)}{%
15   b \sample \bin \\\
16   (\pk,\sk) \sample \kgen(\secparam) \\\
17   (m_0,m_1) \sample \adv^O(\secparam, \pk) \\\
18   c \sample \enc(\pk,m_b) \\\
19   b' \sample \adv(\secparam, \pk, c) \\\
20   \pcreturn b = b' }
21
22 \end{pchstack}
23
24 % alternatively use \pcvspace for spacing
25
26 \begin{pchstack}[space=0.25em]
27 \procedure[linenumbering,mode=text]{Oracle \$O}{
28   Some code \\\
29   Some more code
30 }
31
32 \procedure[linenumbering,mode=text]{Oracle \$H_1}{
33   Some code \\\

```

```

34     Some more code
35 }
36
37 \procedure[linenumbers,mode=text]{Oracle $H_2$}{
38     Some code \\
39     Some more code
40 }
41 \end{pchstack}
42 \end{pcvstack}

```

3.7.1 Stacking Options

The following keys are available on both `pchstack` and `pcvstack` environments

center Centers the stack.

boxed Draws a box around the stack.

space Controls the space between two pseudocode blocks within a stack. The default is 0pt which can be adapted globally by redefining `\pchstackspace` or `\pcvstackspace`.

noindent Does not indent the stack. Only applies if option **center** is not used.

inline Ensures that no paragraph is added by `pchstack`. This cannot be used together with either **center** or **noindent**.

aboveskip By default the outer most stack adds vertical space above. The default space added is `\abovedisplayskip` and can be adapted by redefining `\pcaboveskip`.

belowskip By default the outer most stack adds vertical space below. The default space added is `\belowdisplayskip` and can be adapted by redefining `\pcbelowskip`. Note that the default space below will not be added when used in a floating environment such as a figure. However, when manually setting `belowskip` it will always be added.

3.8 Default Arguments

You can set the default arguments to be used with pseudocode blocks via `\pcsetargs`. This is especially handy in stacking environments to add arguments to all enclosed code blocks.

Some Procedure A	Some Procedure B	Some Procedure C
1 : Step 1	1 : Step 1	1 : Step 1
2 : Step 2	2 : $\left(\begin{smallmatrix} A \\ B+C \end{smallmatrix} \right)$	2 : Step 2
	3 : Step 3	

```

1 \begin{pchstack}[space=1em,center,boxed]
2 % Do not change size to scriptsize for line numbers
3 \renewcommand\pclnstyle[1]{#1}
4
5 % set default arguments for all pseudocode blocks in this hstack

```

```

6 \pcsetargs{linenumbering ,mode=text ,minlineheight=1cm, codesize=\Large{}}
7
8 \procedure{Some Procedue A}{
9   Step 1\\
10  Step 2 }
11
12 \procedure{Some Procedue B}{
13   \text{Step 1}\\
14   \scriptsize$\begin{pcmbbox}\begin{pmatrix}A \\ B + C \end{pmatrix}\end{pcmbbox}
15   }$\\
16   \text{Step 3}}
17
18 \procedure{Some Procedue C}{
19   Step 1\\
20   Step 2 }
\end{pchstack}

```

Default Arguments for Stacking

Similarly to `\pcsetargs` you can define default arguments for `hstack` and `vstack` environments via `\pcsethstackargs` and `\pcsetvstackargs`.

3.9 Divisions and Linebreaks

Within the pseudocode command you generate linebreaks as `\\`. In order to specify the linewidth you can add an optional argument

```
1 \\[height]
```

Furthermore, you can add horizontal lines by using the second optional argument and write

```
1 \\[[hline]
```

$\text{IND-CPA}_{\text{Enc}}^A(1^n)$
1 : $b \leftarrow \$ \{0, 1\}$
<hr style="border: none; border-top: 1px solid black; margin: 10px 0;"/>
2 : $(pk, sk) \leftarrow \$ \text{KGen}(1^n)$
3 : $(m_0, m_1) \leftarrow \$ \mathcal{A}^O(1^n, pk)$
4 : $c \leftarrow \$ \text{Enc}(pk, m_b)$
5 : $b' \leftarrow \$ \mathcal{A}(1^n, pk, c)$
6 : return $b = b'$

```

1 \procedureblock[linenumbering]{\indcpa_{enc}^{adv}(\secparam)}{
2   b \sample \bin \\[2\baselineskip][\hline\hline]
3   (\pk,\sk) \sample \kgen(\secparam) \\
4   (m_0,m_1) \sample \adv^O(\secparam, \pk) \\
5   c \sample \enc(\pk,m_b) \\
6   b' \sample \adv(\secparam, \pk, c) \\
7   \pcreturn b = b' }

```

3.9.1 Optimizing Layout

In case you are laying out multiple procedures horizontally, procedures may be slightly misaligned if the procedure headings are not of the same height. As an example, Consider the following setup

Procedure A	Procedure $B_{G_1}^{F^h*}$
1: do	1: do
2: some	2: some
3: work	3: work

Here the sub and double superscripts in Procedure B make the header slightly larger than the maximum allotted space provided for headers which causes procedure B to be slightly shifted to the bottom. The best way to remedy such a situation is to use a combination of the `headheight` and `headlinesep` properties to increase the header space in both procedures and shift back the headline for a more compact visualization. As we here want to set some arguments for all procedure blocks within the stacking environment we can use `\pcsetargs`.

Procedure A	Procedure $B_{G_1}^{F^h*}$
1: do	1: do
2: some	2: some
3: work	3: work

```

1 \begin{pchstack}[center,space=1ex]
2   \pcsetargs{headheight=5ex,headlinesep=-1ex}
3
4   \procedure[linenumbering]{Procedure $A$}{
5     \text{do}\\
6     \text{some} \\
7     \text{work}
8   }
9
10  \procedure[linenumbering]{Procedure $B^{F^h*}_{G_1}$}{
11    \text{do}\\
12    \text{some} \\
13    \text{work}
14  }
15 \end{pchstack}

```

3.10 Matrices and Math Environments within Pseudocode

In order to work its magic, cryptocode (in particular within the `\pseudocode` command) mingles with a few low level commands such as `\\` or `\halign`. The effect of this is, that when you use certain math environments, for example, to create matrices, within pseudocode the result may be unexpected. Consider the following example

```

1 \pseudocodeblock{
2   \text{compute } P = \begin{pmatrix} A \\ B + C \end{pmatrix}
3   A \\ B + C
4   \end{pmatrix}
5 }

```

which, somewhat unexpectedly, yields

$$\text{compute } P = \begin{pmatrix} A \\ B + C \end{pmatrix}$$

Here, the alignment is somewhat off. In order, to allow for the *pmatrix* environment to properly work without interference from `\pseudocode` you can wrap it into a `pcmbbox` environment (where `pcmbbox` is short for pseudocode math box). This ensures that the low-level changes introduced by `\pseudocode` are not active.

```

1 \pseudocodeblock{
2 \text{compute } P = \begin{pcmbbox}\begin{pmatrix}
3 A \\ B + C
4 \end{pmatrix}\end{pcmbbox}
5 }

```

$$\text{compute } P = \begin{pmatrix} A \\ B + C \end{pmatrix}$$

3.11 Fancy Code with Overlays

Consider the IND-CPA game. Here we have a single adversary \mathcal{A} that is called twice, first to output two messages and which is then given the ciphertext of one of the messages in order to “guess” which one was encrypted. Often this is not visualized. Sometimes an additional state `state` is passed as we have in the following example on the left. On the right, we visualize the same idea in a slightly more fancy way.

IND-CPA _{Enc} ^A (1 ⁿ)	IND-CPA _{Enc} ^A (1 ⁿ)
1: $b \leftarrow \{0, 1\}$	1: $b \leftarrow \{0, 1\}$
2: $(pk, sk) \leftarrow \text{KGen}(1^n)$	2: $(pk, sk) \leftarrow \text{KGen}(1^n)$
3: $(state, m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$	3: $(m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$
4: $c \leftarrow \text{Enc}(pk, m_b)$	4: $c \leftarrow \text{Enc}(pk, m_b)$
5: $b' \leftarrow \mathcal{A}(1^n, pk, c, state)$	5: $b' \leftarrow \mathcal{A}(1^n, pk, c, state)$
6: return $b = b'$	6: return $b = b'$

The image on the right is generated by:

```

1 \begin{pcimage}
2 \procedureblock[linenumbering]{\indcpa_enc^adv(\secpam)}{
3 b \sample \bin
4 (\pk,\sk) \sample \kgen (\secpam)
5 (m_0,m_1) \sample \adv(\secpam, \pk, c) \pcnode{start}
6 c \sample \enc(\pk,m_b)
7 b' \sample \adv(\secpam, \pk, c, \state) \pcnode{end}
8 \pcreturn b = b' }
9
10 \pcdraw{
11 \path[->] (start) edge[bend left=50] node[right]{\state} (start|-end);
12 }
13 \end{pcimage}

```

In order to achieve the above effect cryptocode utilizes TIKZ underneath. The `\pcnode` command generates TIKZ nodes and additionally we wrapped the pseudocode (or procedure) command in an `\begin{pcimage}\end{pcimage}` environment which allows us to utilize these nodes later, for example using the `\pcdraw` command. We can achieve a similar effect without an additional `pcimage` environment by using the optional argument of `\pcnode` for the TIKZ code.

```

1 \procedureblock[linenumbering]{\indcpa_enc^adv(\secpam)}{
2 b \sample \bin
3 (\pk,\sk) \sample \kgen (\secpam)
4 (m_0,m_1) \sample \adv(\secpam, \pk, c) \pcnode{start}

```



```

5 c \sample \enc(\pk,m_b) \\\
6 b' \sample \adv(\secpam, \pk, c, \state) \pcnode{end}[draw={
7 \path[->] (start) edge[bend left=50] node[right]{\state$} (start|-end);
8 }]\\\
9 \pcreturn b = b' }

```

Example: Explain your Code

As an example of what you can do with this, let us put an explanation to a line of the code.

IND-CPA _{Enc} ^A (1 ⁿ)
1: $b \leftarrow \{0, 1\}$
2: $(pk, sk) \leftarrow \text{KGen}(1^n)$
3: $(m_0, m_1) \leftarrow \mathcal{A}(1^n, pk, c)$
4: $c \leftarrow \text{Enc}(pk, m_b)$
5: $b' \leftarrow \mathcal{A}(1^n, pk, c, \text{state})$
6: return $b = b'$

KGen(1ⁿ) samples a public key pk and a private key sk.

```

1 \begin{pcimage}
2 \procedureblock[linenumbering]{\indcpa_\enc^adv(\secpam)}{\%
3 b \sample \bin \\\
4 (\pk,\sk) \sample \kgen (\secpam)\pcnode{kgen} \\\
5 (m_0,m_1) \sample \adv(\secpam, \pk, c) \\\
6 c \sample \enc(\pk,m_b) \\\
7 b' \sample \adv(\secpam, \pk, c, \state) \\\
8 \pcreturn b = b' }
9
10 \pcdraw{
11 \node[rectangle callout,callout absolute pointer=(kgen),fill=orange]
12 at ([shift={(+3,+1)}]kgen) {
13 \begin{varwidth}{3cm}
14 $\kgen(\secpam)$ samples a public key $\pk$ and a private key $\sk$.
15 \end{varwidth}
16 };
17 }
18 \end{pcimage}

```

4 Tabbing Mode

In the following section we discuss how to create multiple columns within a `\pseudocode` command. Within a `\pseudocode` command you can switch to a new column by inserting a `\>`. This is similar to using an `align` environment and placing a tabbing character (`&`). Also, similarly to using `align` you should ensure that the number of `\>` are identical on each line.

First	Second	Third	Fourth
$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$

```

1 \pseudocodeblock{
2   \textbf{First} \> \textbf{Second} \> \textbf{Third} \> \textbf{Fourth} \> \>
3   b \sample \bin \> b \sample \bin \> b \sample \bin \> b \sample \bin}

```

As you can see the first column is left aligned the second right, the third left and so forth. In order to get only left aligned columns you could thus simply always skip a column by using `\>\>`. You can also use `\<` a shorthand for `\>\>`.

First	Second	Third	Fourth
$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$

```

1 \pseudocodeblock{
2   \textbf{First} \< \textbf{Second} \< \textbf{Third} \< \textbf{Fourth} \< \>
3   b \sample \bin \< b \sample \bin \< b \sample \bin \< b \sample \bin}

```

Column Spacing You can control the space between columns using the option “`colsep=2em`”. Note that when doing so you should additionally use “`addtolength=5em`” (where 5em depends on the number of columns) in order to avoid having overfull hboxes.

First	Second	Third	Fourth
$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}$

```

1 \pseudocodeblock[ colsep=1em, addtolength=10em]{%
2   \textbf{First} \< \textbf{Second} \< \textbf{Third} \< \textbf{Fourth} \< \>
3   b \sample \bin \< b \sample \bin \< b \sample \bin \< b \sample \bin}

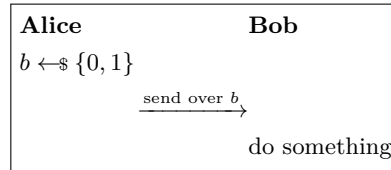
```

This is basically all you need to know in order to go on to writing protocols with the cryptocode package. So unless you want to know a bit more about tabbing (switching columns) and learn some of the internals, feel free to proceed to Section 5.

4.1 Tabbing in Detail

At the heart of the pseudocode package is an `align` (or rather a `flalign*`) environment which allows you to use basic math notation. Usually an `align` (or `flalign`) environment uses `&` as tabbing characters. The pseudocode comes in two modes the first of which changes the default align behavior. That is, it automatically adds a tabbing character to the beginning and end of each line and changes the tabbing character to `\>`. This mode is called *mintabmode* and is active by default.

In mintabmode in order to make use of extra columns in the align environment (which we will use shortly in order to write protocols) you can use `\>` as you would use `&` normally. But, don't forget that there is an alignment tab already placed at the beginning and end of each line. So the following example



is generated by

```

1 \pseudocodeblock{
2   \textbf{Alice} \> \> \textbf{Bob}  \\\
3   b \sample \bin \> \> \\\
4   \> \xrightarrow{\text{send over } b} \>  \\\
5   \> \> \text{do something}}

```

4.1.1 Overriding The Tabbing Character

If you don't like `\>` as the tabbing character you can choose a custom command by overwriting `\pctabname`. For example

```

1 \renewcommand{\pctabname}{\myTab}
2
3 \pseudocode{
4   \textbf{Alice} \myTab \myTab \textbf{Bob}  \\\
5   b \sample \bin \myTab \myTab \\\
6   \myTab \xrightarrow{\text{send over } b} \myTab  \\\
7   \myTab \myTab \text{do something}}

```

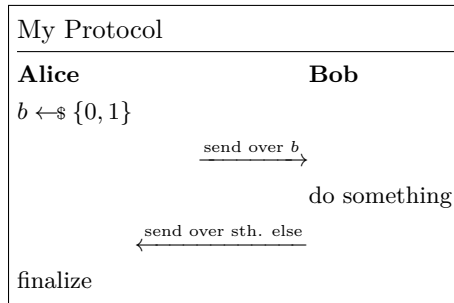
Similarly you can redefine the double tabbing character `\<` by overwriting `\pcdbltabname` (also see Section 5).

4.1.2 Custom Line Spacing and Horizontal Rules

As explained, underlying the pseudocode command is an `flalign` environment. This would allow the use of `\\[spacing]` to specify the spacing between two lines or of `\\[hline]` to insert a horizontal rule. In order to achieve the same effect within the pseudocode command you can use `\\[spacing] [hline]`. You can also use `\pclb` to get a line break which does not insert the additional alignment characters.

5 Protocols

Using tabbing, we can use `\pseudocode` to also layout protocols such as



which is generated as

```

1 \procedureblock{My Protocol}{
2   \textbf{Alice} \> \> \textbf{Bob} \> \>
3   b \sample \bin \> \> \> \>
4   \> \xrightarrow{\text{send over } b} \> \>
5   \> \> \text{do something} \> \>
6   \> \xleftarrow{\text{send over sth. else}} \> \>
7   \text{finalize} \> \>

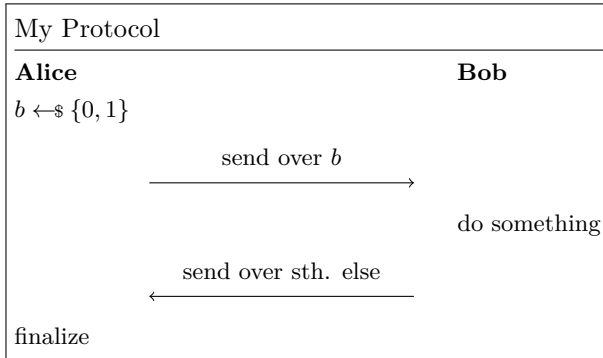
```

In order to get nicer message arrows use the commands `\sendmessageright*{message}`, `\sendmessageleft*{message}`, and `\sendmessagerightleft*{message}`. All three take an additional optional argument specifying the length of the arrow and all wrap their mandatory argument in an `aligned` environment.

```

1 \sendmessageright*[3.5cm]{message}
2 \sendmessageleft*[3.5cm]{message}

```



```

1 \procedureblock{My Protocol}{%
2   \textbf{Alice} \> \> \textbf{Bob} \> \>
3   b \sample \bin \> \> \> \>
4   \> \sendmessageright*{\text{send over } b} \> \>
5   \> \> \text{do something} \> \>
6   \> \sendmessageleft*{\text{send over sth. else}} \> \>
7   \text{finalize} \> \>

```

To obtain granular control over how messages are set use the `\sendmessage` and `\sendmessage*` commands. These take two parameters, the first being the message style for the underlying TIKZ path (e.g., `->` for messages to the right) and the second a key

value list of arguments. The difference between the starred version and the unstarred version is that the starred version wraps its labels in an **aligned** environment. Following is an example, that showcases various message options.

My Protocol

Alice

Bob

$b \leftarrow \{0, 1\}$

send over b

Text below

do something

send over sth. else

a, b, c

c, d, e

foo

1: you can also

2: use pseudocode

foo

finalize

```
1 \procedureblock{My Protocol}{
2   \textbf{Alice} \> \> \textbf{Bob}  \\\
3   b \sample \bin \> \> \\\
4   \> \sendmessage{<->}{centercol=3,top=send over $b$,bottom=Text below,topstyle={
5     draw,solid,yshift=0.25cm},style={dashed}} \> \\\
6   \> \> \text{do something} \\\
7   \> \sendmessage{<->}{length=8cm,top=send over sth. else} \> \\\
8   \> \sendmessage*{<->}{length=8cm,top=\pseudocode[linenumbering]{\text{you can
9     also}}\text{use pseudocode}},bottom={foo}} \> \\\
   \text{finalize} \> \> }
```

sendmessage and **sendmessage*** support the following options:

top The content to display on top of the arrow.

bottom The content to display below the arrow.

left The content to display on the left of the arrow.

right The content to display on the right of the arrow.

topstyle The TIKZ style to be used for the top node.

bottomstyle The TIKZ style to be used for the bottom node.

rightstyle The TIKZ style to be used for the right node.

leftstyle The TIKZ style to be used for the left node.

length The length of the arrow.

style The style of the arrow.

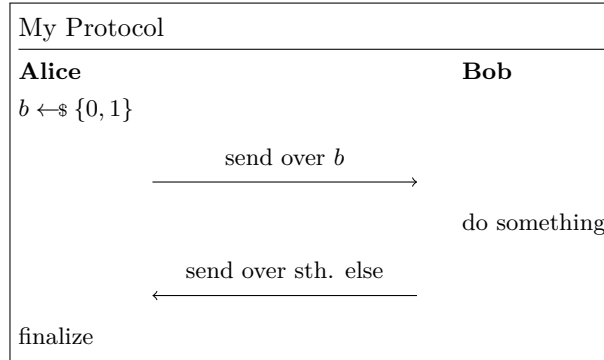
width The width of the column

centercol Can be used to ensure that the message is displayed in the center. This should be set to the column index. In the above example, the message column is the third column (note that there is a column left of alice that is automatically inserted).

5.1 Tabbing

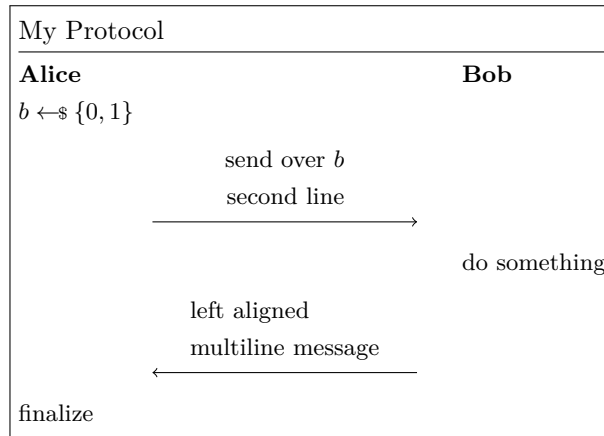
When typesetting protocols you might find that using two tabs instead of a single tab usually provides a better result as this ensures that all columns are left aligned. For this you can use `\<` instead of `\>` (see Section 4).

Following is once more the example from before but now with double tapping.



5.2 Multiline Messages

Using the starred send message commands you can easily generate multiline messages as the command wraps an *aligned* environment around the message.



```

1 \procedureblock{My Protocol}{%
2 \textbf{Alice} \< \< \textbf{Bob} \> \>
3 b \sample \bin \< \< \> \>
4 \< \sendmessageright*{\text{send over } b\> \text{second line}} \< \>
5 \< \< \text{do something} \> \>
6 \< \sendmessage*{\<-\}{top=\>\text{left aligned}} \> \> \text{multiline message}
7 \> \> \> \>
8 \text{finalize} \< \<

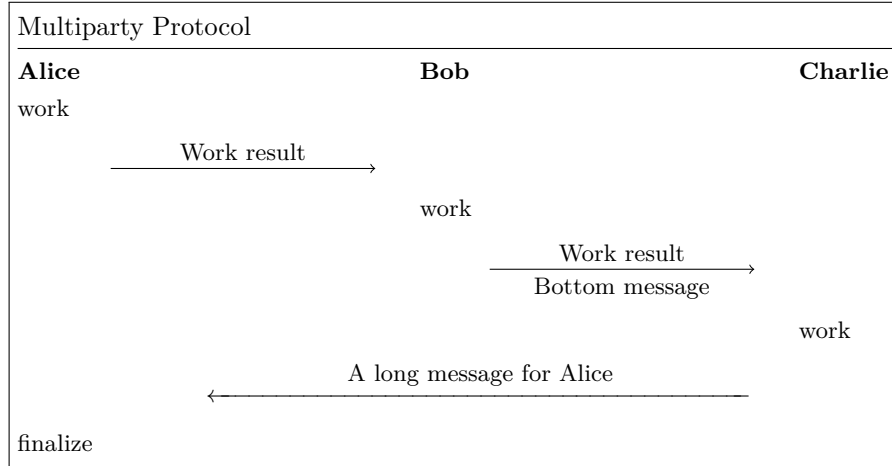
```

Remark. When using `\sendmessage*` the tabbing character `&` cannot be used. Instead use the `\>` command as defined within `\pseudocode`.

5.2.1 Multiplayer Protocols

You are not limited to two players. In order to send messages skipping players use `\sendmessagerightx` and `\sendmessageleftx`.

```
1 \sendmessagerightx[width]{columnspan}{Text}
2 \sendmessageleftx[width]{columnspan}{Text}
```



```
1 \procedureblock{Multiparty Protocol}{%
2 \textbf{Alice} \< \< \textbf{Bob} \< \< \textbf{Charlie} \< \<
3 \text{work} \< \< \< \< \< \<
4 \< \sendmessageright{top=Work result} \< \< \< \< \< \<
5 \< \< \text{work} \< \< \< \<
6 \< \< \< \sendmessageright{top=Work result ,bottom=Bottom message} \< \<
7 \< \< \< \< \text{work} \< \< \< \<
8 \< \sendmessageleftx[7cm]{8}{\text{A long message for Alice}} \< \<
9 \text{finalize} \< \< \< \< }
```

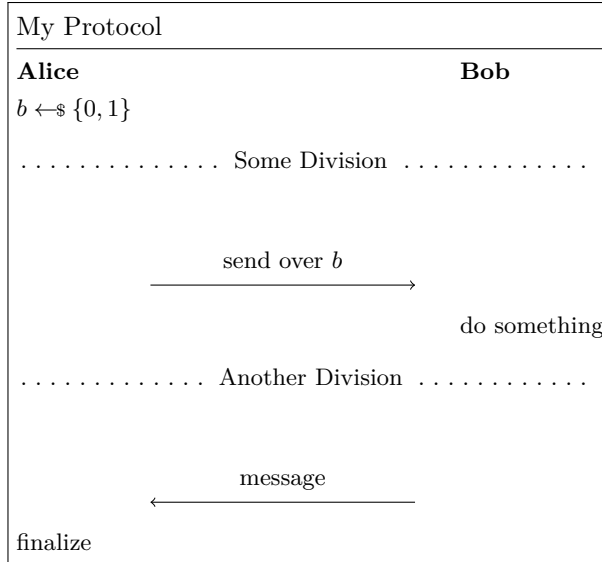
Note that for the last message from Charlie to Alice we needed to specify the number of passed over columns (`\sendmessageleftx[7cm]{8}{message}`). As we were passing 4 `\<` where each creates 2 columns, the total was 8 columns.

5.2.2 Divisions

You can use `\pcintertext` in order to divide protocols (or other pseudocode for that matter).

```
1 \pcintertext[dotted|center]{Division Text}
```

Note that in order to use the `\pcintertext` you need to use `\pclb` as the line break for the line before. Also see Section 4.



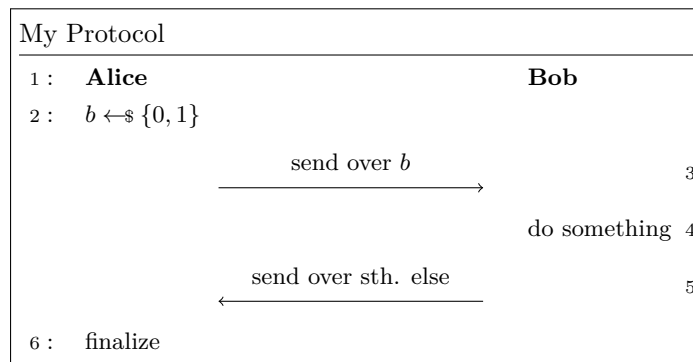
```

1 \procedureblock{My Protocol}{%
2   \textbf{Alice} \< \< \textbf{Bob} \< \<
3   b \sample \bin \< \< \pclb
4   \pcintertext[dotted]{Some Division} \< \<
5   \< \sendmessageright*{\text{send over } b} \< \<
6   \< \< \text{do something} \< \< \pclb
7   \pcintertext[dotted]{Another Division} \< \<
8   \< \sendmessageleft*{\text{message}} \< \<
9   \text{finalize} \< \< }

```

5.3 Line Numbering in Protocols

Protocols can be numbered similarly to plain pseudocode. Additionally to the `\pcln` there are the commands `\pclnr` and `\pcrln`. The first allows you to right align line numbers but uses the same counter as `\pcln`. The second uses a different counter.



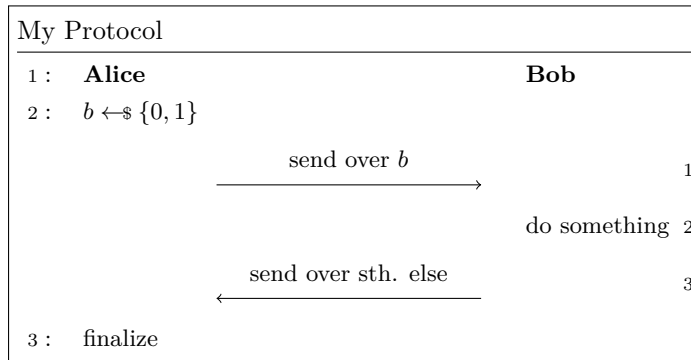
Which is generated as

```

1 \procedureblock{My Protocol}{
2   \pcln \textbf{Alice} \< \< \textbf{Bob} \< \<
3   \pcln b \sample \bin \< \< \< \<
4   \< \sendmessageright*{\text{send over } b} \< \< \pclnr \<
5   \< \< \text{do something} \< \< \pclnr \<
6   \< \sendmessageleft*{\text{send over sth. else}} \< \< \pclnr \<
7   \pcln \text{finalize} \< \< \< }

```


And using `\pcrln` we obtain:



This is generated as

```

1 \procedureblock{My Protocol}{%
2 \pcln \textbf{Alice} \< \< \textbf{Bob} \> \>
3 \pcln b \sample \bin \< \< \> \>
4 \< \sendmessageright*{\text{send over } b} \< \pcrln\>
5 \< \< \text{do something} \pcrln \>
6 \< \sendmessageleft*{\text{send over sth. else}} \< \pcrln \>
7 \pcln \text{finalize} \< \< }

```

5.3.1 Separators

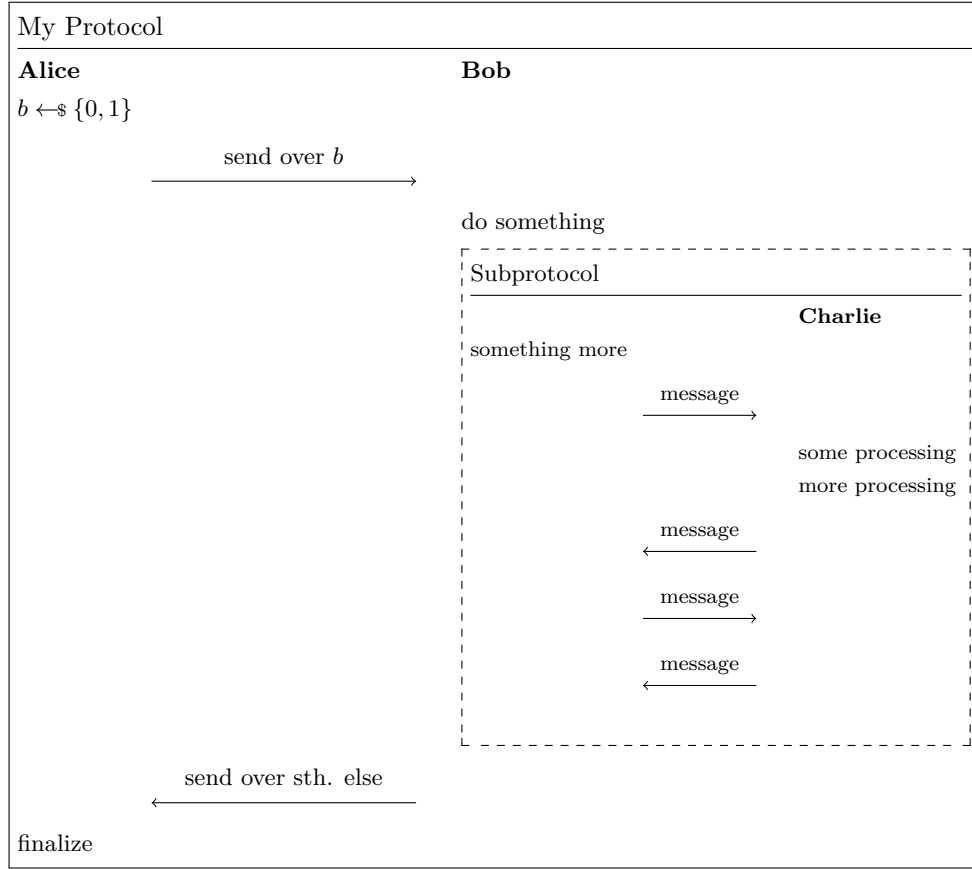
The commands `\pclnseparator` and `\pcrlnseparator` define the separators between code and line number. By default the left separator is set to `(:)` colon and the right separator is set to an empty string.

5.3.2 Spacing

Spacings after the left separator and in front of the right separator can be controlled by `\pclnspace` and `\pclnrspace` which are set to 1em and 0.5em, respectively.

5.4 Sub Protocols

Use the `subprocedure` environment to also create sub protocols.



```

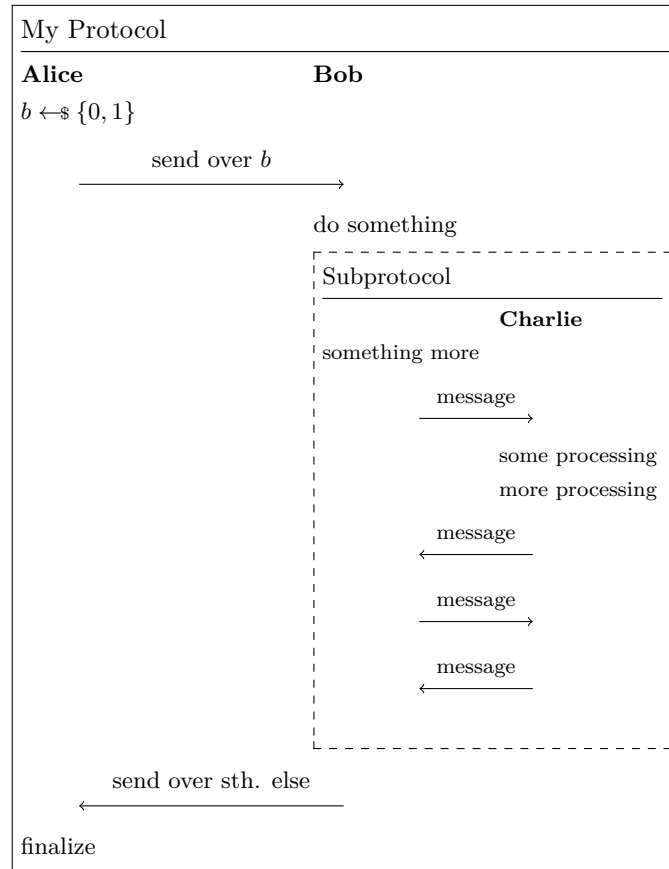
1 \procedureblock{My Protocol}{
2   \textbf{Alice} \< \< \textbf{Bob} \> \>
3   b \sample \bin \< \< \> \>
4   \< \< \sendmessageright*{\text{send over } b} \< \< \> \>
5   \< \< \text{do something} \> \>
6   \< \< \dbox{\begin{subprocedure}\procedure{Subprotocol}{
7     \< \< \textbf{Charlie} \> \>
8     \text{something more} \< \< \> \>
9     \< \< \sendmessageright*[1.5cm]{\text{message}} \< \< \> \>
10    \< \< \text{some processing} \> \>
11    \< \< \text{more processing} \> \>
12    \< \< \sendmessageleft*[1.5cm]{\text{message}} \< \< \> \>
13    \< \< \sendmessageright*[1.5cm]{\text{message}} \< \< \> \>
14    \< \< \sendmessageleft*[1.5cm]{\text{message}} \< \< \> \>
15    \end{subprocedure}} \> \>
16    \< \< \sendmessageleft*{\text{send over sth. else}} \< \< \> \>
17    \text{finalize} \< \< \> \>

```

5.5 Compact Presentation of Protocols

In order to present protocols more compactly you can use the `colspace` option which adds space inbetween two columns. When set to a negative space, this has the effect that columns overlap. The following example is once more our above example using a

sub protocol but this time with `colspace=-1cm`. Note that the sub protocol inherits the option which is why both the outer and the inner protocol now have overlapping columns.



```

1 \procedureblock[colspace=-1cm]{My Protocol}{
2   \textbf{Alice} \< \< \textbf{Bob} \> \>
3   b \sample \bin \< \< \> \>
4   \< \sendmessageright*{\text{send over } b} \< \> \>
5   \< \< \text{do something} \> \>
6   \< \< \dbox{\begin{subprocedure}\procedure{Subprotocol}{
7     \< \< \textbf{Charlie} \> \>
8     \text{something more} \< \< \> \>
9     \< \sendmessageright*[1.5cm]{\text{message}} \< \> \>
10    \< \< \text{some processing} \> \>
11    \< \< \text{more processing} \> \>
12    \< \sendmessageleft*[1.5cm]{\text{message}} \< \> \>
13    \< \sendmessageright*[1.5cm]{\text{message}} \< \> \>
14    \< \sendmessageleft*[1.5cm]{\text{message}} \< \> \>
15    }\end{subprocedure}} \> \>
16    \< \sendmessageleft*{\text{send over sth. else}} \< \> \>
17    \text{finalize} \< \< \> \>

```

6 Game-Based Proofs

6.1 Basics

Besides displaying pseudocode the package also comes with commands to help presetn game-based proofs. The `gameproof` environment wraps the pseudocode block of a game-based proof.

```
1 \begin{gameproof}
2   proof goes here
3 \end{gameproof}
```

Within a `gameproof` environment use the command `\gameprocedure` which works similarly to the pseudocode command and produces a heading of the form $\text{Game}_{\text{counter}}(n)$ where counter is a consecutive counter. Thus, we can create the following setup

$\text{Game}_1(n)$	$\text{Game}_2(n)$
1 : Step 1	Step 1
2 : Step 2	Step 2

by using

```
1 \begin{gameproof}
2 \begin{pchstack}[space=1em,center,boxed]
3 \gameprocedure[linenumbering,mode=text]{%
4   Step 1  \\
5   Step 2
6 }
7 \gameprocedure[mode=text]{%
8   Step 1  \\
9   Step 2
10 }
11 \end{pchstack}
12 \end{gameproof}
```

For discussing individual games, cryptocode provides the `\pcgame` command which without argument prints `Game` and with (optional) argument `\pcgame[n]` prints `Gamen`.

6.1.1 Highlight Changes

In order to highlight changes from one game to the next use `\gamechange`.

$\text{Game}_1(n)$	$\text{Game}_2(n)$
1 : Step 1	Step 1
2 : Step 2	Step 2

```
1 \begin{gameproof}
2 \begin{pchstack}[space=1em,center,boxed]
3 \gameprocedure[linenumbering,mode=text]{%
4   Step 1  \\
5   Step 2
6 }
7 \gameprocedure[mode=text]{%
8   Step 1  \\
9   \gamechange{Step 2}
10 }
11 \end{pchstack}
12 \end{gameproof}
```

The background color can be controlled by redefining `\gamechange color` which by default is defined as

```
1 \definecolor{gamechange color}{gray}{0.90}
```

Remark. Note that `\gamechange` is always in text mode.

6.1.2 Boxed Games

Use `\tbxgameprocedure` in order to create two consecutive games where the second game is *boxed*. Use `\pcbox` to create boxed statements.

$\text{Game}_1(n)$	$\text{Game}_2(n)$	$\text{Game}_3(n)$	$\text{Game}_4(n)$
1 : Step 1	Step 1;	Alternative step 1	Step 1
2 : Step 2	Step 2 is different		Step 2

```
1 \begin{gameproof}
2 \begin{pchstack}[space=1em,boxed,center]
3 \gameprocedure[linenumbering]{
4   \text{Step 1} \quad \quad
5   \text{Step 2}
6 }
7 \tbxgameprocedure{
8   \text{Step 1}; \pcbox{\text{Alternative step 1}} \quad \quad
9   \gamechange{\text{Step 2 is different}}
10 }
11 \gameprocedure{
12   \text{Step 1} \quad \quad
13   \text{\gamechange{Step 2}}
14 }
15 \end{pchstack}
16 \end{gameproof}
```

6.1.3 Reduction Hints

In a game based proof, in order to go from one game to the next we usually give a reduction, for example, we show that the difference between two games is bound by the security of some pseudorandom generator PRG. To give a hint within the pseudocode that the difference between two games is down to “something” you can use the `\addgamehop` command.

```
1 \addgamehop{startgame}{endgame}{options}
```

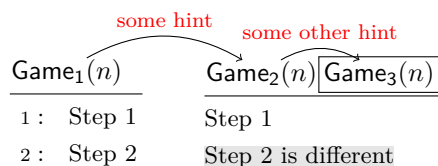
Here options allows you to specify the hint as well as the style. The following options are available

hint The hint text

nodestyle A TIKZ style to be used for the node.

pathstyle A TIKZ style to be used for the path.

edgestyle A TIKZ style to be used for the edge. This defaults to “bend left”.



```

1 \begin{gameproof}
2 \begin{pchstack}[center,space=2em]
3   \gameprocedure[linenumbering]{
4     \text{Step 1}   \\
5     \text{Step 2}
6   }
7   \tbxgameprocedure{
8     \text{Step 1}   \\
9     \gamechange{\text{Step 2 is different}}
10  }
11 \end{pchstack}
12
13 \addgamehop{1}{2}{hint=\footnotesize some hint,nodestyle=red}
14 \addgamehop{2}{3}{hint=\footnotesize some other hint}
15 \end{gameproof}

```

The edgestyle allows you to specify how the hint is displayed. If you, for example want a straight line, rather than the curved arrow simply use

```

1 \addgamehop{1}{2}{hint=\footnotesize some hint,edgestyle=}

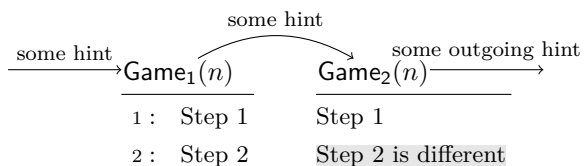
```

If game proofs do not fit into a single picture you can specify start and end hints using the commands

```

1 \addstartgamehop[first game]{options}
2 \addendgamehop[last game]{options}

```



```

1 \begin{gameproof}
2 \begin{pchstack}[center,space=2em]
3   \gameprocedure[linenumbering]{
4     \text{Step 1}   \\
5     \text{Step 2}
6   }
7   \gameprocedure{
8     \text{Step 1}   \\
9     \gamechange{\text{Step 2 is different}}
10  }
11 \end{pchstack}
12
13 \addstartgamehop{hint=\footnotesize some hint,edgestyle=}
14 \addgamehop{1}{2}{hint=\footnotesize some hint}
15 \addendgamehop{hint=\footnotesize some outgoing hint,edgestyle=}
16 \end{gameproof}

```

6.1.4 Numbering and Names

By default the `gameproof` environment starts to count from 1 onwards. Its optional parameters allow you to specify a custom name for the game as well as defining the starting number.

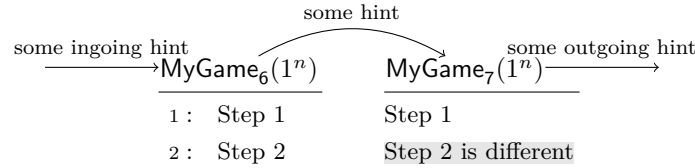
```
1 \begin{gameproof}[options]
```

The following parameters are available which, as usual, are provided in a key=value based form.

nr The starting number minus 1. Thus, when setting nr=5, the first game will be Game_6 .

name The name for the game

arg The argument to be used for the game.



```

1 \begin{gameproof}[nr=5,name=\mathsf{MyGame},arg=(1^n)]
2 \begin{pchstack}[center,space=2em]
3   \gameprocedure[linenumbering]{
4     \text{Step 1} \quad \backslash
5     \text{Step 2} \quad \}
6   \gameprocedure{
7     \text{Step 1} \quad \backslash
8     \gamechange{\text{Step 2 is different}} \}
9 \end{pchstack}
10
11 \addstartgamehop{hint=\footnotesize some ingoing hint,edgestyle=}
12 \addgamehop{6}{7}{hint=\footnotesize some hint}
13 \addendgamehop{hint=\footnotesize some outgoing hint,edgestyle=}
14 \end{gameproof}

```

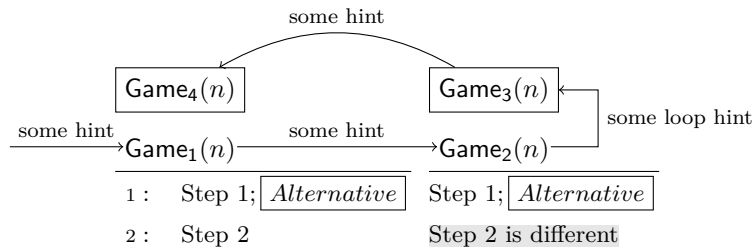
6.1.5 Default Name and Argument

The default name and argument are controlled via the commands `\pcgamenam` and `\gameprocedurearg`.

Command	Default
<code>\pcgamenam</code>	<code>\mathsf{Game}</code>
<code>\gameprocedurearg</code>	<code>\secpa</code>

6.1.6 Bi-Directional Games

You can use the `\bxgameprocedure` to generate games for going in two directions. Use the `\addloopgamehop` to add the gamehop in the middle.



```

1 \begin{gameproof}
2 \bxgameprocedure{4}{%
3 \pcin \text{Step 1}; \pcbox{Alternative} \\
4 \pcin \text{Step 2}
5 }
6 \bxgameprocedure{3}{%
7 \text{Step 1}; \pcbox{Alternative} \\
8 \gamechange{\text{Step 2 is different}}
9 }
10 \addstartgamehop{hint=\footnotesize some hint,edgestyle=}
11 \addgamehop{1}{2}{hint=\footnotesize some hint,edgestyle=}
12 \addloopgamehop{hint=\footnotesize some loop hint}
13 \addgamehop{2}{1}{hint=\footnotesize some hint}
14 \end{gameproof}

```

6.1.7 Styling Game Procedures

It may come in handy to define default style arguments for the underlying pseudocode command used by `\gameprocedure`. For this you can define the default arguments by calling `\setgameproceduredefaultstyle` to for example:

```

1 \setgameproceduredefaultstyle{beginline=\vphantom{\bin^{\prg_\prg}}

```

The default is to not set any options.

6.2 Game Descriptions

Cryptocode also comes with an environment to provide textual descriptions of games such as

reduction target
 \downarrow **MyGame₃(n)**: This is the third game. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis condimentum velit et orci volutpat, sed ultrices lorem lobortis. Nam vehicula, justo eu varius interdum, felis mi consectetur dolor, ac posuere nulla lacus varius diam. Etiam dapibus blandit leo, et porttitor augue lacinia auctor.

MyGame₄(n): This is the fourth game. The arrow at the side indicates the reduction target.

The above example is generated as

```

1 \begin{gamedescription}[name=MyGame,nr=2]
2 \describegame
3 This is the third game. Lorem ipsum dolor sit amet, consectetur adipiscing elit
. Duis condimentum velit et orci volutpat, sed ultrices lorem lobortis. Nam
vehicula, justo eu varius interdum, felis mi consectetur dolor, ac posuere
nulla lacus varius diam. Etiam dapibus blandit leo, et porttitor augue
lacinia auctor.
4
5 \describegame[inhint=reduction target]
6 This is the second game. The arrow at the side indicates the reduction target.
7 \end{gamedescription}

```

The `gamedescription` environment takes an optional argument to specify name and counter (defaults to Game and 0). The command `\describegame` starts a new game description and can allow you to provide a reduction hint using the option parameter `inhint`.

inhint Displays an ingoing arrow to denote the reduction target for this game hop.

length Allows to control the length of the arrow.

nodestyle Allows to control the style of the node.

hint Instead of having an ingoing arrow, this adds an outgoing arrow.

7 Black-Box Reductions

The cryptocode package comes with support for drawing basic black box reductions. A reduction always takes the following form.

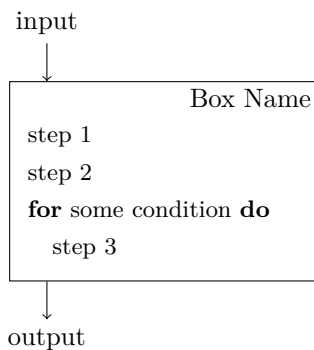
```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Box Name]
3 % The Box's content
4 \end{bbrbox}
5 % Commands to display communication, input output etc
6 \end{bbrenv}

```

That is, a **bbrenv** environment (where bbr is short for black-box reduction) which takes a single **bbrbox** environment plus some additional commands.

Following is a simple example with a single (black)box and some code plus inputs outputs:



This box is generated as

```

1 \begin{bbrenv}[aboveskip=1cm,belowskip=1cm]{A}
2 \begin{bbrbox}[name=Box Name]
3 \pseudocode{
4   \text{step 1} \\
5   \text{step 2} \\
6   \pcfor \text{some condition} \pcdo \\
7   \t\text{step 3}
8 }
9 \end{bbrbox}
10 \bbrinput{input}
11 \bbroutput{output}
12 \end{bbrenv}

```

The commands **bbrinput** and **bbroutput** allow to specify input and output for the latest **bbrenv** environment. The optional parameter for the **bbrenv** environment allows to specify leading and trailing space (this may become necessary when using inputs and outputs). For this provide **aboveskip** and **belowskip** keys. (Note that in an earlier version of cryptocode you could write `\begin{bbrenv}[1cm]{A}[1cm]`. While this format is still supported it should be regarded deprecated.)

The **bbrenv** environment takes the following options as optional first parameter:

aboveskip Space above.

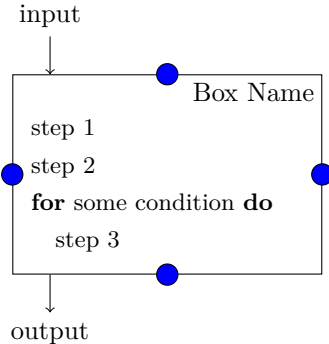
belowskip Space below.

tikzargs Underneath **bbrenv** is a **tikzpicture** and via **tikzargs** you can pass in arguments.

The single mandatory argument to the `bbrenv` environment needs to specify a unique identifier (unique for the current reduction). This id is used as an internal TIKZ node name (<https://www.ctan.org/pkg/pgf>).

```
1 \begin{bbrenv}[options]{UNIQUE IDENTIFIER}
2 % deprecated version
3 \begin{bbrenv}[vspace before]{UNIQUE IDENTIFIER}[vspace after]
```

As we are drawing a TIKZ image, note that we can easily later customize the image using the labels that we have specified on the way.



```
1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Box Name]
3 \pseudocode{
4   \text{step 1} \\
5   \text{step 2} \\
6   \pcfor \text{some condition} \pcdo \\
7   \pcind \text{step 3}
8 }
9 \end{bbrbox}
10 \bbrinput{input}
11 \bbroutput{output}
12
13 \filldraw[fill=blue] (A.north) circle (4pt);
14 \filldraw[fill=blue] (A.west) circle (4pt);
15 \filldraw[fill=blue] (A.east) circle (4pt);
16 \filldraw[fill=blue] (A.south) circle (4pt);
17 \end{bbrenv}
```

The `bbrbox` takes as single argument a comma separated list of key value pairs. In the example we used

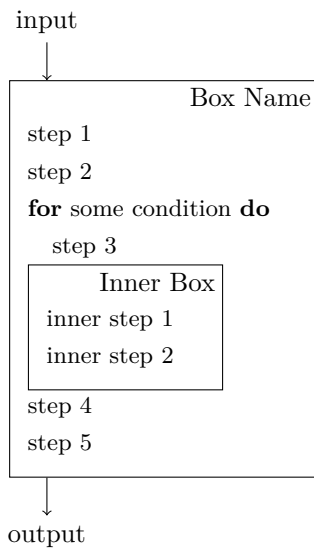
```
1 name=Box Name
```

to specify the label. The following options are available

Option	Description
name	Specifies the box's label
namepos	Specifies the position (left, center, right, top left, top center, top right, middle)
namestyle	Specifies the style of the name
abovesep	Space above box (defaults to <code>\baselineskip</code>)
minheight	The minimal height
addheight	Additional height at the end of the box
xshift	Allows horizontal positioning
yshift	Allows horizontal positioning
style	allows to customize the node

7.1 Nesting of Boxes

Boxes can be nested. For this simply insert a `bbrenv` (together with a single `bbrbox`) environment into an existing `bbrbox`.



```

1  \begin{bbrenv}{A}
2  \begin{bbrbox}[name=Box Name]
3  \pseudocode{
4    \text{step 1} \\
5    \text{step 2} \\
6    \pcfor \text{some condition} \pcdo \\
7    \pcind \text{step 3}
8  }
9
10 \begin{bbrenv}{B}
11 \begin{bbrbox}[name=Inner Box]
12 \pseudocode{
13   \text{inner step 1} \\
14   \text{inner step 2}
15 }
16 \end{bbrbox}
17 \end{bbrenv}
18
19 \pseudocode{
20   \text{step 4} \\
21   \text{step 5}
22 }
23 \end{bbrbox}
24 \bbrinput{input}
25 \bbroutput{output}
26 \end{bbrenv}

```

7.2 Messages and Queries

You can send messages and queries to boxes. For this use the commands

```

1 \bbrmsgto{options}
2 \bbrmsgfrom{options}
3 \bbrmsgtofrom{options}{options}
4 \bbrmsgfromto{options}{options}
5 \bbrqryto{options}
6 \bbrqryfrom{options}
7 \bbrqrytofrom{options}{options}
8 \bbrqryfromto{options}{options}

```

By convention messages are on the left of boxes and queries on the right. Commands ending on **to** make an arrow to the right while commands ending on **from** make an arrow to the left. The **options** define how the message is drawn and consists of a key-value list. The **tofrom** and **fromto** variants draw two messages (back and forth) that are more compactly set together. Here usually, the first message should be drawn on top (**top=Label**) while the second message should be drawn on the bottom (**bottom=Label**).

For example, to draw a message with a label on top and on the side use

```

1 \bbrmsgto{top=Top Label, side=Side Label}

```

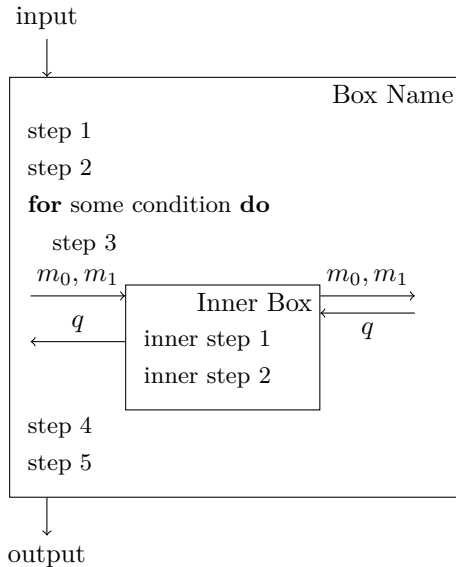
If your label contains a “,” (comma), then group the label in {} (curly brackets).

```

1 \bbrmsgto{top=Top Label, side={Side, Label}}

```

Following is a complete example. Notice that cryptocode takes care of the vertical positioning.



```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Box Name]
3 \pseudocode{

```

```

4      \text{step 1} \\
5      \text{step 2} \\
6      \pcfor \text{some condition} \pcdo \\
7      \pcind \text{step 3}
8    }
9
10   \begin{bbrenv}{B}
11     \begin{bbrbox}[name=Inner Box]
12       \pseudocode{
13         \text{inner step 1} \\
14         \text{inner step 2}
15       }
16     \end{bbrbox}
17
18     \bbrmsgto{top={$m_0,m_1$}}
19     \bbrmsgfrom{top=$q$}
20
21     \bbrqrytofrom{top={$m_0,m_1$}}{bottom=$q$}
22
23   \end{bbrenv}
24
25   \pseudocode{
26     \text{step 4} \\
27     \text{step 5}
28   }
29 \end{bbrbox}
30 \bbrinput{input}
31 \bbroutput{output}
32 \end{bbrenv}

```

7.2.1 Options

Following is a list of all available options. Remember that underneath the reduction commands is a TIKZ image (<https://www.ctan.org/pkg/pgf/>) and for each label position (top, side, bottom) a node is generated which can be further customized via low-level TIKZ.

top Label on top

bottom Label on the bottom

side Label on the far side of the box. For challengers and oracles, on the side of the box.

oside Label on the “other” side.

topstyle Style for label on top

bottomstyle Style for label on bottom

sidestyle Style for label on side

osidestyle Style for label on other side

edgestyle Style for edge

length Length of arrow

topname Name for node on top

bottomname Name for node on bottom

sidenname Name for node on side

osidenname Name for node on other side

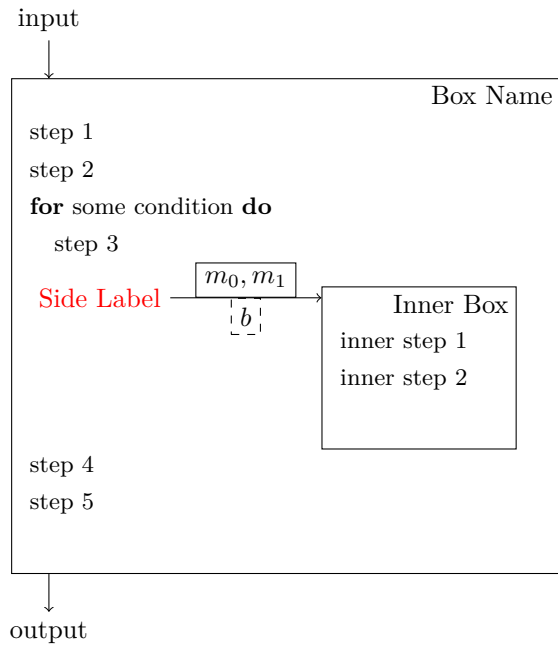
aboveskip Space before message

belowskip Space after message

fixedoffset Ignores automatic spacing and sets the message at the provided offset from the top.

fixedboffset Ignores automatic spacing and sets the message at the provided offset from the bottom.

islast Places the message at the bottom.



```
1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Box Name]
3 \pseudocode{
4 \text{step 1} \\
5 \text{step 2} \\
6 \pcfor \text{some condition} \pcdo \\
7 \pcind \text{step 3}
8 }
9
10 \begin{bbrenv}{B}
11 \begin{bbrbox}[name=Inner Box]
12 \pseudocode{
13 \text{inner step 1} \\
14 \text{inner step 2} \\
15 }
16 \end{bbrbox}
17
18 \bbrmsgto{top={\$m_0,m_1$},side=Side Label, bottom=$b$, length=2cm,
19 topstyle={draw, solid}, sidestyle={red}, bottomstyle={draw, dashed}}
20
21 \end{bbrenv}
22
23 \pseudocode{
24 \text{step 4} \\
25 \text{step 5} \\
26 }
27 \end{bbrbox}
28 \bbrinput{input}
```

```

29 \bbroutput{output}
30 \end{bbrenv}

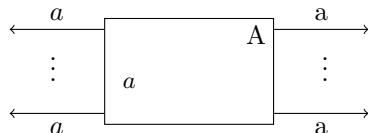
```

First Message

The first message is offset by `\bbrfirstmessageoffset` which defaults to `1ex`.

7.2.2 Vdots

You can use `\bbrmsgvdots` and `\bbrqryvdots` to add `\vdots` in between messages or queries.



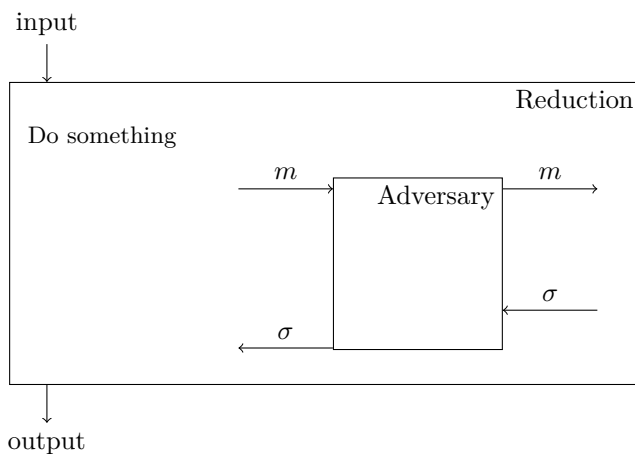
```

1 \begin{bbrenv}{A}
2   \begin{bbrbox}[name=A,minheight=14mm]
3     \pseudocode{a}
4   \end{bbrbox}
5   \bbrmsgfrom{top={$a$}}
6   \bbrmsgvdots
7   \bbrmsgfrom{islast=true,bottom={$a$}}
8   \bbrqryto{top=a}
9   \bbrqryvdots%
10  \bbrqryto{bottom=a,islast=true}
11 \end{bbrenv}

```

7.2.3 Add Space

If the spacing between messages is not sufficient you can use the `\bbrmsgspace` and `\bbrqryspace` commands to add additional space. Alternatively, you can use the options `aboveskip` and `belowskip` on the individual message or query commands.



```

1 \begin{bbrenv}{A}
2   \begin{bbrbox}[name=Reduction]
3     \pseudocode{
4       \text{Do something}
5     }
6

```

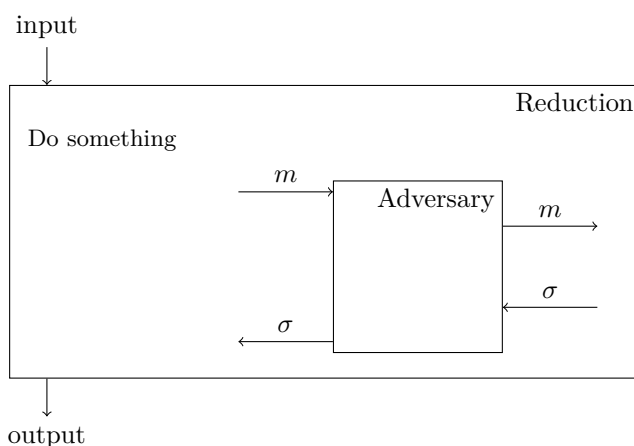


```

7 \begin{bbrenv}{B}
8
9   \begin{bbrbox}{name=Adversary , minheight=15ex , xshift=4cm}
10
11   \end{bbrbox}
12
13   \bbrmsgto{top=$m$}
14   \bbrmsgspace{1.5cm}
15   \bbrmsgfrom{top=$\sigma$}
16
17   \bbrqryto{top=$m$}
18   \bbrqryspace{1cm}
19   \bbrqryfrom{top=$\sigma$}
20
21 \end{bbrenv}
22
23 \end{bbrbox}
24 \bbrinput{input}
25 \bbroutput{output}
26 \end{bbrenv}

```

Note that for placing a message at the bottom, `islast` or fixed offsets often allow obtain more accurate results.



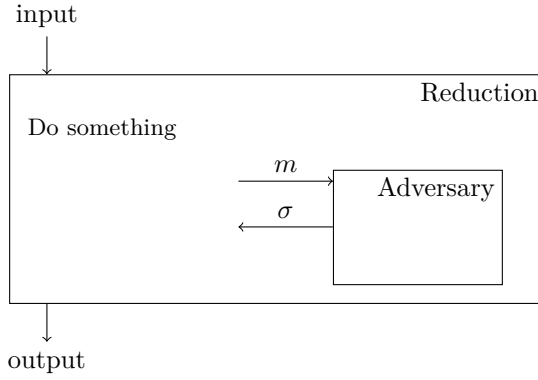
```

1 \begin{bbrenv}{A}
2   \begin{bbrbox}{name=Reduction}
3     \pseudocode{
4       \text{Do something}
5     }
6
7     \begin{bbrenv}{B}
8
9       \begin{bbrbox}{name=Adversary , minheight=15ex , xshift=4cm}
10
11       \end{bbrbox}
12
13       \bbrmsgto{top=$m$}
14       \bbrmsgfrom{top=$\sigma$, islast}
15
16       \bbrqryto{top=$m$, fixedoffset=4ex}
17       \bbrqryfrom{top=$\sigma$, fixedboffset=4ex}
18
19     \end{bbrenv}
20
21   \end{bbrbox}
22   \bbrinput{input}
23   \bbroutput{output}
24 \end{bbrenv}

```

7.2.4 Loops

Often an adversary may send poly many queries to an oracle, or a reduction sends many queries to an adversary. Consider the following setting



```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Reduction]
3 \pseudocode{
4   \text{Do something}
5 }
6
7 \begin{bbrenv}{B}
8
9   \begin{bbrbox}[name=Adversary , minheight=10ex , xshift=4cm]
10
11   \end{bbrbox}
12
13   \bbrmsgto{top=$m$}
14   \bbrmsgfrom{top=$\sigma$}
15
16 \end{bbrenv}
17
18 \end{bbrbox}
19 \bbrinput{input}
20 \bbroutput{output}
21 \end{bbrenv}

```

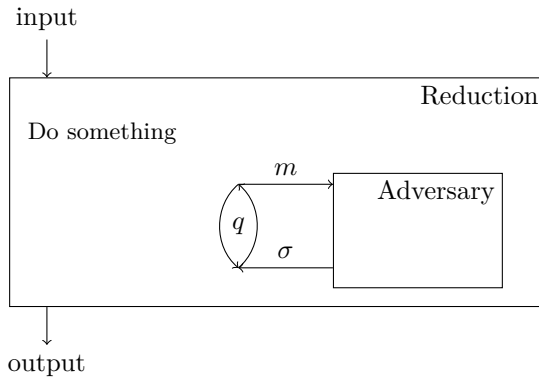
First note that by specifying the minheight and xshift option we shifted the adversary box a bit to the right and enlarged its box. Further we specified custom names for the node on the side of the two messages. We can now use the **bbrloop** command to visualize that these two messages are exchanged q many times

```

1 \bbrloop{BeginLoop}{EndLoop}{center=$q$}

```

The **bbrloop** command takes two node names and a config which allows you to specify if the label is to be shown on the left, center or right. Here is the result.



```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Reduction]
3 \pseudocode{
4   \text{Do something}
5 }
6
7 \begin{bbrenv}{B}
8
9   \begin{bbrbox}[name=Adversary ,minheight=10ex ,xshift=4cm]
10
11   \end{bbrbox}
12
13   \bbrmsgto{top=$n$,sidename=BeginLoop}
14   \bbrmsgspace{0.5cm}
15   \bbrmsgfrom{top=$\sigma$,sidename=EndLoop}
16   \bbrloop{BeginLoop}{EndLoop}{center=$q$}
17
18 \end{bbrenv}
19
20 \end{bbrbox}
21 \bbrinput{input}
22 \bbroutput{output}
23 \end{bbrenv}

```

The `\bbrloop` command supports the following parameters:

- center** Label displayed within the loop
- left** Label displayed left of the loop
- right** Label displayed right of the loop
- centerstyle** Style for center label
- leftstyle** Style for left label
- rightstyle** Style for right label
- clockwise** Loop going in clockwise direction
- angle** Angle of the arrows

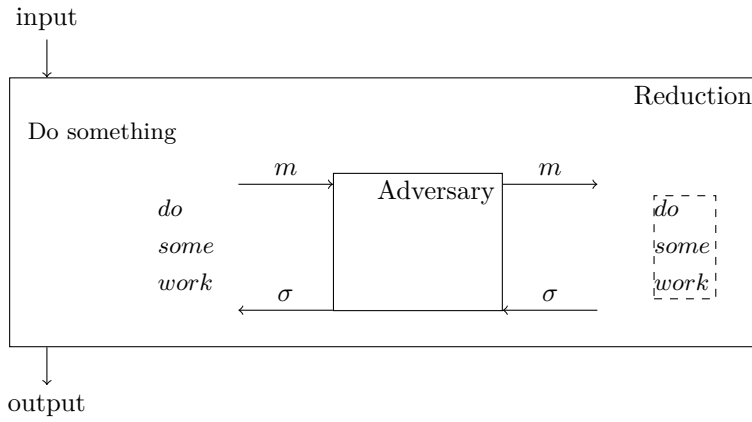
7.2.5 Intertext

If your reduction needs to do some extra work between queries use the `\bbrmsgtxt` and `\bbrqrytxt` commands.

```

1 \bbrmsgtxt[options]{Text}
2 \bbrqrytxt[options]{Text}

```



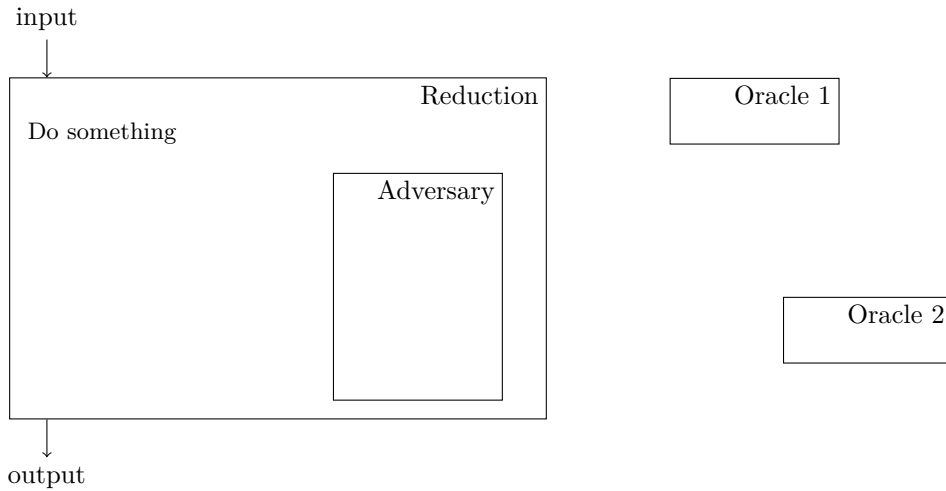
```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Reduction]
3 \pseudocode{
4   \text{Do something}
5 }
6
7 \begin{bbrenv}{B}
8
9   \begin{bbrbox}[name=Adversary ,minheight=12ex ,xshift=4cm]
10
11   \end{bbrbox}
12
13   \bbrmsgto{top=$m$}
14   \bbrmsgtxt{\pseudocode{
15     do \\
16     some \\
17     work
18   }}
19   \bbrmsgfrom{top=$\sigma$}
20
21   \bbrqryto{top=$m$}
22   \bbrqrytxt[nodestyle={draw,dashed},xshift=2cm]{\pseudocode{
23     do \\
24     some \\
25     work
26   }}
27   \bbrqryfrom{top=$\sigma$}
28
29 \end{bbrbox}
30
31 \end{bbrenv}
32 \bbrinput{input}
33 \bbroutput{output}
34 \end{bbrenv}

```

7.3 Oracles

Each box can have one or more oracles which are drawn on the right hand side of the box. An oracle is created similarly to a **bbrenv** environment using the **bbroracle** environment. Oracles go behind the single **bbrbox** environment within an **bbrenv** environment.



```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Reduction]
3 \pseudocode{
4 \text{Do something}
5 }
6
7 \begin{bbrenv}{B}
8 \begin{bbrbox}[name=Adversary,minheight=3cm,xshift=4cm]
9 \end{bbrbox}
10
11 \end{bbrenv}
12
13 \end{bbrbox}
14 \bbrinput{input}
15 \bbroutput{output}
16
17 \begin{bbroracle}{OraA}
18 \begin{bbrbox}[name=Oracle 1]
19 \end{bbrbox}
20 \end{bbroracle}
21
22 \begin{bbroracle}{OraB}[vdistance=2cm,hdistance=3cm]
23 \begin{bbrbox}[name=Oracle 2]
24 \end{bbrbox}
25 \end{bbroracle}
26 \end{bbrenv}

```

Via the option “`hdistance=length`” and “`vdistance=length`” you can control the horizontal and vertical position of the oracle. By default this value is set to 1.5cm and `\baselineskip`.

7.3.1 Communicating with Oracles

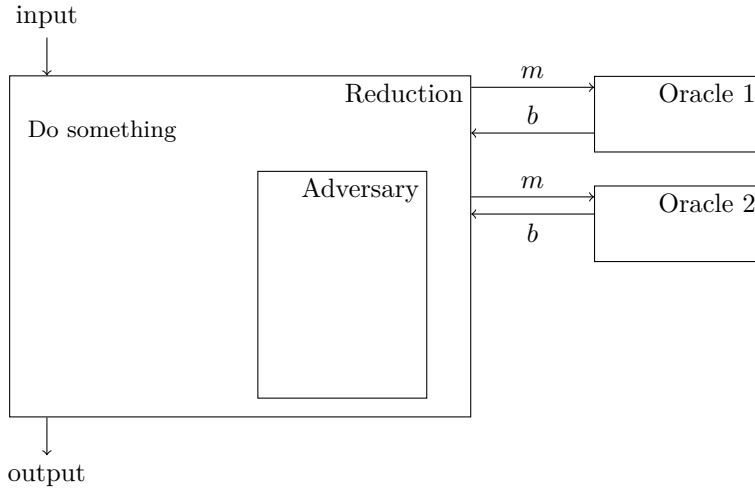
As oracles use the `bbrbox` environment we can directly use the established ways to send messages and queries to oracles. In addition you can use the `\bbroracleqryfrom` and `\bbroracleqryto`.

```

1 \bbroracleqryfrom{options}
2 \bbroracleqryto{options}
3 \bbroracleqrytofrom{options}{options}
4 \bbroracleqryfromto{options}{options}

```

Here options allow you to specify where the label goes (top, bottom). In addition you can use `\bbroracleqryspspace` to generate extra space between oracle messages. Note that oracle messages need to be added after the closing `\end{bbroracle}` command.



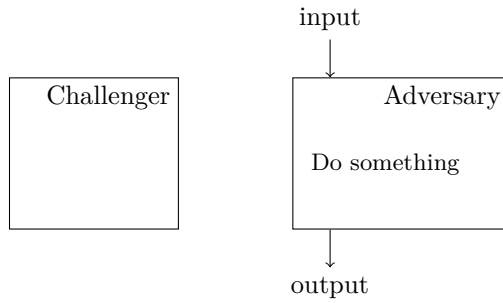
```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Reduction]
3 \pseudocode{
4 \text{Do something}
5 }
6
7 \begin{bbrenv}{B}
8 \begin{bbrbox}[name=Adversary,minheight=3cm,xshift=3cm]
9 \end{bbrbox}
10
11 \end{bbrenv}
12
13 \end{bbrbox}
14 \bbrinput{input}
15 \bbroutput{output}
16
17 \begin{bbroracle}{OraA}
18 \begin{bbrbox}[name=Oracle 1,minheight=1cm]
19 \end{bbrbox}
20 \end{bbroracle}
21 \bbroracleqrytotop={m$}
22 \bbroracleqryfrom{top=b$}
23
24 \begin{bbroracle}{OraB}
25 \begin{bbrbox}[name=Oracle 2,minheight=1cm]
26 \end{bbrbox}
27 \end{bbroracle}
28 \bbroracleqrytofrom{top={m$}}{bottom=b$}
29 \end{bbrenv}

```

7.4 Challengers

Each box can have one or more challengers which are drawn on the left hand side of the box. Challengers behave identically to oracles with the exception that they are to the left of the box. A challenger is created similarly to a *bbrenv* environment using the *bbrchallenger* environment. Challengers go behind the single *bbrbox* environment within an *bbrenv* environment.



```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Adversary ,minheight=2cm]
3 \pseudocode{
4   \text{Do something}
5 }
6
7 \end{bbrbox}
8 \bbrinput{input}
9 \bbroutput{output}
10
11 \begin{bbrchallenger}{ChaA}
12   \begin{bbrbox}[name=Challenger ,minheight=2cm]
13
14   \end{bbrbox}
15 \end{bbrchallenger}
16 \end{bbrenv}

```

Via the option “hdistance=length” and “vdistance=length” you can control the horizontal and vertical position of the challenger. By default this value is set to 1.5cm and `\baselineskip`.

7.4.1 Communicating with Challengers

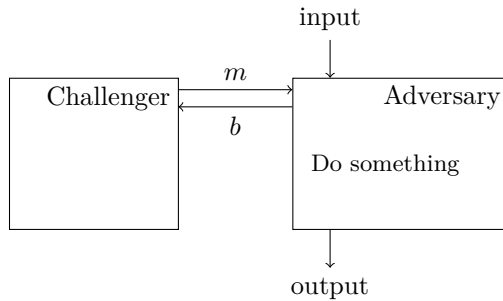
As challengers use the `bbrbox` environment we can directly use the established ways to send messages and queries to oracles. In addition you can use the `\bbrchallengerqryfrom` and `\bbrchallengerqryto`.

```

1 \bbrchallengerqryfrom{options}
2 \bbrchallengerqryto{options}
3 \bbrchallengerqrytofrom{options}{options}
4 \bbrchallengerqryfromto{options}{options}

```

Here options allow you to specify where the label goes (top, bottom). In addition you can use `\bbrchallengerqryspspace` to generate extra space between oracle messages. Note that challenger messages need to be added after the closing `\end{bbrchallenger}` command.



```

1 \begin{bbrenv}{A}
2 \begin{bbrbox}[name=Adversary ,minheight=2cm]

```

```

3 \pseudocode{
4   \text{Do something}
5 }
6
7 \end{bbrbox}
8 \bbrinput{input}
9 \bbroutput{output}
10
11 \begin{bbrchallenger}{ChaA}
12   \begin{bbrbox}[name=Challenger ,minheight=2cm]
13
14   \end{bbrbox}
15 \end{bbrchallenger}
16
17 \bbrchallengerqryfromto{top=$n$}{bottom=$b$}
18 \end{bbrenv}

```

7.5 Horizontal Stacking

`bbrenv` environments can be stacked horizontally.



Note that in order to not have horizontal space inbetween the boxes, that you need to not leave any space. In the following code this is handled via comments.

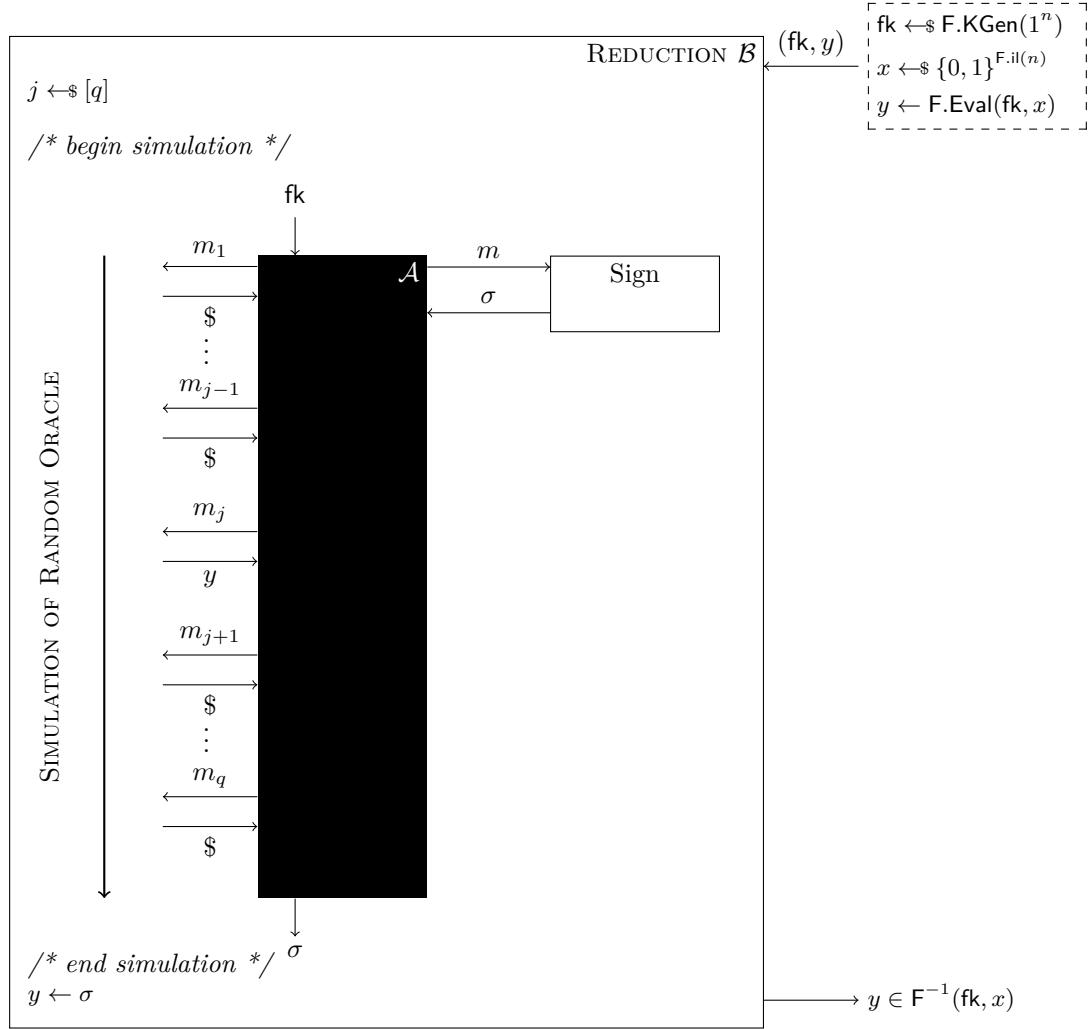
```

1 \begin{figure}[h]
2   \centering
3   \begin{bbrenv}{A}
4     \begin{bbrbox}[name=A,minheight=15mm]
5       \pseudocode{a}
6       \end{bbrbox}
7       \bbrmsgfrom{islast=true,top={$a$}}
8       \bbrqryto{aboveskip=6mm,top={$2a$}}
9     \end{bbrenv}%
10  \begin{bbrenv}{B}
11    \begin{bbrbox}[name=B,minheight=15mm]
12      \end{bbrbox}
13      \bbrqryto{edgestyle={<->}}
14      \bbrqryvdots[aboveskip=1mm]
15      \bbrqryto{islast=true,edgestyle={<->}}
16    \end{bbrenv}%
17  \begin{bbrenv}{C}
18    \begin{bbrbox}[name=C,minheight=15mm]
19      \end{bbrbox}
20      \bbrqryto{top={$3a$}}
21    \end{bbrenv}%
22 \end{figure}

```

7.6 Examples

A reduction sketch for full domain hash.



```

1 \begin{bbrenv}{Red}
2
3 \begin{bbrbox}[name=\textsc{Reduction }]\bdv$]
4
5 \pseudocode{
6   j \sample [q]
7 }
8
9 \vspace{2ex}
10 \emph{/* begin simulation */}
11
12 \begin{bbrenv}[aboveskip=2em]{Adv}
13   \begin{bbrbox}[name=\adv$,minheight=8.5cm,style={fill=black},namestyle={
14     color=white},xshift=3cm]
15     \end{bbrbox}
16
17     \bbrinput{\$fk\$}
18     \bbroutput{\$\sigma\$}
19
20     \bbrmsgfrom{top=\$m_1$,afterskip=-0.5\baselineskip}
21     \bbrmsgto{bottom=\$\\$,afterskip=0.5\baselineskip}
22
23     \bbrmsgvdots
24
25     \bbrmsgfrom{top=\$m_{j-1}$,beforekip=0.5\baselineskip,afterskip=-0.5\
26     \baselineskip}
27     \bbrmsgto{bottom=\$\\$,afterskip=1.5\baselineskip}

```

```

26 \bbrmsgfrom{top=$n_j$,afterskip=-0.5\baselineskip}
27 \bbrmsgto{bottom=$y$,afterskip=1.5\baselineskip}
28
29
30 \bbrmsgfrom{top=$n_{j+1}$,afterskip=-0.5\baselineskip}
31 \bbrmsgto{bottom=$\$$,afterskip=0.5\baselineskip}
32
33 \bbrmsgvdots
34
35 \bbrmsgfrom{top=$n_q$,beforeskip=0.5\baselineskip,afterskip=-0.5\
baselineskip}
36 \bbrmsgto{bottom=$\$$}
37
38 \begin{bbroracle}{Sign}
39 \begin{bbrbox}[name=Sign,namepos=center,style={draw},minheight=1cm]
40 \end{bbrbox}
41 \end{bbroracle}
42
43 \bbroracleqryto{top=$n$}
44 \bbroracleqryfrom{top=$\sigma$}
45
46 \end{bbrenv}
47
48 \pcdraw{
49 \node[left=2cm of Adv.north west] (startsim) {};
50 \node[left=2cm of Adv.south west] (endsim) {};
51 \draw[->,thick] (startsim) — (endsim);
52 \node[rotate=90, left=2.75cm of Adv.west,anchor=center] () {\textsc{
Simulation of Random Oracle}};
53 }
54
55 \emph{/* end simulation */}
56
57 \pseudocode{
58 y \gets \sigma
59 }
60
61 \end{bbrbox}
62 \bbrqryfrom{beforeskip=0.25cm,top={$(\text{fk}, y)$},side={\dbox{\pseudocode{
63 \fk \sample \fash.\kgen(\seccparam) \ x \sample \bin^{\fash.\il(\seccpar)} \ \
y \gets \fash.\eval(\text{fk}, x)}
64 }}}
65 \bbrqryto{beforeskip=11.75cm,side=\pseudocode{y \in \fash^{-1}(\text{fk}, x)}}
66 \end{bbrenv}

```

8 Known Issues

8.1 Pseudocode KeepSpacing within Commands

The (experimental) “space=keep” option of pseudocode which should output spacing identical to that of the input will fail, if the pseudocode command is called from within another command. An example is to wrap the `\pseudocode` command in an `\fbox` or in a stacking environment such as `\pchstack`. As a workaround for generating frame boxes you should hence use a package such as *mdframed* (<https://www.ctan.org/pkg/mdframed>) which provides a frame environment.

	Pseudocode	with	- spaces -	
1	<code>\pseudocode[space=keep,mode=text]{</code>	Pseudocode	with	
	<code>- spaces -}</code>			

As an alternative you could use a *savebox* (in combination with the `lrbox` environment):

	Pseudocode	with	- spaces -	
1	<code>\newsavebox{\mypcbox}</code>			
2	<code>\begin{lrbox}{\mypcbox}%</code>			
3	<code>\pseudocode[space=keep,mode=text]{</code>	Pseudocode	with	
	<code>- spaces -}%</code>			
4	<code>\end{lrbox}</code>			
5	<code>\fbox{\usebox{\mypcbox}}</code>			

8.2 AMSFonts

Some packages are not happy with the “amsfonts” package. Cryptocode will attempt to load amsfonts if it is loaded with either the “sets” or the “probability” option. In order to not load amsfonts you can additionally add the “noamsfonts” at the very end. Note that in this case you should ensure that the command `\mathbb` is defined as this is used by most of the commands in “sets” and some of the commands in “probability”.

8.3 Hyperref

The hyperref package (<https://www.ctan.org/pkg/hyperref>) should be loaded before cryptocode. If this is not possible call the `\pcfixhyperref` after `\begin{document}`.

8.4 Cleveref

In order to support the cleveref package (<https://ctan.org/pkg/cleveref>) load cleveref after cryptocode and subsequently (but still in the preamble) call `\pcfixcleveref`.

8.5 Babel - Spanish

The spanish version of the babel package uses `<` and `>` as shorthands which are used by cryptocode as tabbing characters. The easiest workaround is to tell cryptocode to use different tabbing characters, for example:

```
1 \renewcommand{\pctabname}{ctab}  
2 \renewcommand{\pcdbltabname}{cdtab}
```

9 Implementation

Following is the implementation of cryptocode. The source code documentation is a work in progress.

```
1 (*cryptocode.sty)
```

Note that most macros are prefixed with *pc* short for pseudocode. This is a general design choice to not conflict with macros defined by other packages. One exception are the macros defined via the various package options.

Load amsmath and mathtools early on, before defining various macros.

```
2 \RequirePackage{amsmath}
3 \RequirePackage{mathtools}
```

9.1 Package Options

`\@pc@opt@amsfonts` Definitions of boolean flags used to determin whether or not to load amsfonts.

```
4 \newif\if@pc@opt@amsfonts
```

`\@pc@opt@advantage` Whether or not to define commands for the given option.

```
5 \newif\if@pc@opt@advantage
```

`\@pc@opt@centernot` Whether or not to load centernot

```
6 \newif\if@pc@opt@centernot
```

9.1.1 operators

`\sample` Definitions of macros for the *operators* pacakge option.

```
\floor      7 \DeclareOption{operators}{
\tfloor      \tfloor      7 \DeclareOption{operators}{
\ceil        \ceil        \tfloor      7 \DeclareOption{operators}{
```

Robust sample operator that also works in subscripts. Is based on egreg's solution given in <https://tex.stackexchange.com/questions/418740/how-to-write-left-arrow-with-a-dollar-sign>

```
\Angle      8 \providecommand\sample{\leftarrow\mathrel{\mkern-2.0mu}\pc@smalldollar}
\tAngle     9 \newcommand{\pc@smalldollar}{\mathrel{\mathpalette\pc@small@dollar\relax}}
\abs       10 \newcommand{\pc@small@dollar}[2]{%
\tabs      11 \vcenter{\hbox{%
\norm     12   $#1\textnormal{\fontsize{0.7\dimexpr\fontdimen2\textfont2}{0}\selectfont\hskip-0.05em plus 0.5em}%
\tnorm    13   }}%
\concat    14 }
\emptystring 15
\argmax    16 \DeclarePairedDelimiter\pc@floor{\lfloor}{\rfloor}
\argmin    17 \providecommand{\floor}[1]{\pc@floor*{#1}}
\pindist   18 \providecommand{\tfloor}[1]{\pc@floor{#1}}
\cindist   19
\sindist   20 \DeclarePairedDelimiter\pc@ceil{\lceil}{\rceil}
          21 \providecommand{\ceil}[1]{\pc@ceil*{#1}}
          22 \providecommand{\tceil}[1]{\pc@ceil{#1}}
          23
          24 \DeclarePairedDelimiter\pc@Angle{\langle}{\rangle}
          25 \providecommand{\Angle}[1]{\pc@Angle*{#1}}
          26 \providecommand{\tAngle}[1]{\pc@Angle{#1}}
          27
          28 \DeclarePairedDelimiter\pc@abs{\lvert}{\rvert}
          29 \providecommand{\abs}[1]{\pc@abs*{#1}}
          30 \providecommand{\tabs}[1]{\pc@abs{#1}}
```

```

31
32 \DeclarePairedDelimiter\pc@norm{\lVert}{\rVert}
33 \providecommand{\norm}[1]{\pc@norm*{#1}}
34 \providecommand{\tnorm}[1]{\pc@tnorm*{#1}}
35
36 \providecommand{\concat}{\ensuremath{\| \|}}
37 \providecommand{\emptystring}{\ensuremath{\varepsilon}}
38
39 \DeclareMathOperator*{\argmax}{arg\,max}
40 \DeclareMathOperator*{\argmin}{arg\,min}
41
42 %indistinguishability
43 \newcommand{\@pc@oset}[3][0ex]{%
44   \mathrel{\mathop{#3}\limits^{
45     \vbox to#1{\kern-2\ex@
46       \hbox{$\scriptstyle#2$}\vss}}}}
47
48 \newcommand{\pindist}{\@pc@oset{\text{p}}{\lower.2ex\hbox{$=$}}}
49 \newcommand{\sindist}{\@pc@oset{\text{s}}{\lower.1ex\hbox{$\approx$}}}
50 \newcommand{\cindist}{\@pc@oset{\text{c}}{\lower.1ex\hbox{$\approx$}}}
51 }

```

9.1.2 adversary

`\adversary` Definitions of adversaries \mathcal{A} (`\adv`), \mathcal{B} (`\bdv`), etc. together with a style `\pcadvstyle`.

```

\adv    52 \DeclareOption{adversary}{
\bdv    53 \providecommand{\adversary}[1]{\pcadvstyle{#1}}
\cdv    54
\ddv    55 \providecommand{\adv}{\pcadvstyle{A}}
\edv    56 \providecommand{\bdv}{\pcadvstyle{B}}
\mdv    57 \providecommand{\cdv}{\pcadvstyle{C}}
\pdv    58 \providecommand{\ddv}{\pcadvstyle{D}}
\rdv    59 \providecommand{\edv}{\pcadvstyle{E}}
\sdv    60 \providecommand{\mdv}{\pcadvstyle{M}}
        61 \providecommand{\pdv}{\pcadvstyle{P}}
        62 \providecommand{\rdv}{\pcadvstyle{R}}
        63 \providecommand{\sdv}{\pcadvstyle{S}}
        64 }

```

9.1.3 landau

`\bigO` Defines several *Landau symbols*.

```

\smallO 65 \DeclareOption{landau}{
\bigOmega 66 \providecommand{\bigO}[1]{\ensuremath{\mathcal{O}\pc@olrk*{#1}}}
\smallOmega 67 \providecommand{\smallO}[1]{\ensuremath{\text{O}\pc@olrk*{#1}}}
\bigsmallO 68 \providecommand{\bigOmega}[1]{\ensuremath{\Omega\pc@olrk*{#1}}}
\bigTheta 69 \providecommand{\smallOmega}[1]{\ensuremath{\omega\pc@olrk*{#1}}}
\orderOf 70 \providecommand{\bigsmallO}[1]{%
        71 \PackageWarning{cryptocode}{bigsmallO is deprecated. Use bigTheta instead.}%
        72 \ensuremath{\Theta\pc@olrk*{#1}}}
        73 \providecommand{\bigTheta}[1]{\ensuremath{\Theta\pc@olrk*{#1}}}
        74 \providecommand{\orderOf}{\ensuremath{\sim}}
        75 }

```

9.1.4 probability

`\probnam` The *probability* package option defines various macros for typesetting probabilities.

`\expectationname` Sets flags `\@pc@opt@amsfontstrue`.

`\supportname` 76 `\DeclareOption{probability}{`

`\tprob` 77 `\@pc@opt@amsfontstrue`

`\prob` 78

`\tprobsub` 79 `\providecommand{\probnam}{Pr}`

`\probsub` 80 `\providecommand{\expectationname}{\ensuremath{\mathbb{E}}}`

`\probsublong` 81 `\providecommand{\supportname}{Supp}`

`\tcondprob` 82

`\condprob` 83 `\providecommand{\tprob}[1]{\ensuremath{\operatorname{\probnam}\pc@elrk{#1}}}`

`\tcondprobsub` 84 `\providecommand{\prob}[1]{\ensuremath{\operatorname{\probnam}\pc@elrk*{#1}}}`

`\condprobsub` 85

`\texpect` 86 `\providecommand{\tprobsub}[2]{\ensuremath{\operatorname{\probnam}_{#1}\pc@elrk{#2}}}`

`\expect` 87 `\providecommand{\probsub}[2]{\ensuremath{\operatorname{\probnam}_{#1}\pc@elrk*{#2}}}`

`\texpsub` 88 `\providecommand{\probsublong}[2]{\ensuremath{\prob{#2}\, , : \, , #1}}`

`\expsub` 89

`\tcondexp` 90 `\providecommand{\tcondprob}[2]{\ensuremath{\tprob{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

`\condexp` 91 `\providecommand{\condprob}[2]{\ensuremath{\prob{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

`\tcondexpsub` 92

`\condexpsub` 93 `\providecommand{\tcondprobsub}[3]{\ensuremath{\tprobsub{#1}{#2}\, \left| \, , #3\, \vphantom{#1}\right. }}`

`\supp` 94 `\providecommand{\condprobsub}[3]{\ensuremath{\probsub{#1}{#2}\, \left| \, , #3\, \vphantom{#1}\right. }}`

`\entropy` 95

`\condentropy` 96 `\providecommand{\texpect}[1]{\ensuremath{\operatorname{\expectationname}\pc@elrk{#1}}}`

`\minentropy` 97 `\providecommand{\expect}[1]{\ensuremath{\operatorname{\expectationname}\pc@elrk*{#1}}}`

`\condminentropy` 98

`\condavgminentropy` 99 `\providecommand{\texpsub}[2]{\ensuremath{\operatorname{\expectationname}_{#1}\pc@elrk{#2}}}`

100 `\providecommand{\expsub}[2]{\ensuremath{\operatorname{\expectationname}_{#1}\pc@elrk*{#2}}}`

101

102 `\providecommand{\tcondexp}[2]{\ensuremath{\texpect{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

103 `\providecommand{\condexp}[2]{\ensuremath{\expect{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

104

105 `\providecommand{\tcondexpsub}[3]{\ensuremath{\texpsub{#1}{#2}\, \left| \, , #3\, \vphantom{#1}\right. }}`

106 `\providecommand{\condexpsub}[3]{\ensuremath{\expsub{#1}{#2}\, \left| \, , #3\, \vphantom{#1}\right. }}`

107

108 `\providecommand{\supp}[1]{\ensuremath{\operatorname{Supp}\pc@olrk*{#1}}}`

109

110 `\providecommand{\entropy}[1]{\ensuremath{\operatorname{H}\pc@olrk*{#1}}}`

111 `\providecommand{\condentropy}[2]{%`

112 `\ensuremath{\operatorname{H}\pc@olrk*{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

113

114 `\providecommand{\minentropy}[1]{\ensuremath{\operatorname{H}_{\infty}\pc@olrk*{#1}}}`

115 `\providecommand{\tminentropy}[1]{\ensuremath{\operatorname{H}_{\infty}\pc@olrk*{#1}}}`

116 `\providecommand{\condminentropy}[2]{%`

117 `\ensuremath{\operatorname{H}_{\infty}\pc@olrk*{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

118 `\providecommand{\tcondminentropy}[2]{%`

119 `\ensuremath{\operatorname{H}_{\infty}\pc@olrk*{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

120 `\providecommand{\condavgminentropy}[2]{%`

121 `\ensuremath{\operatorname{\tilde{H}_{\infty}\pc@olrk*{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

122 `\providecommand{\tcondavgminentropy}[2]{%`

123 `\ensuremath{\operatorname{\tilde{H}_{\infty}\pc@olrk*{#1}\, \left| \, , #2\, \vphantom{#1}\right. }}`

124 }

9.1.5 sets

`\NN` The *sets* option defines various macros for standard sets such as natural numbers `\NN`
`\ZZ` (\mathbb{N}). The style can be configured via `\pcsetstyle`.
`\CC` As we usually work with bit strings, the macro `\bin` defines the set $\{0,1\}$. Sets the
`\QQ` flags `\@pc@opt@amsfontstrue`.
`\RR` 125 `\DeclareOption{sets}{`
`\PP` 126 `\@pc@opt@amsfontstrue`
`\FF` 127
`\GG` 128 `\providecommand\NN{\pcsetstyle{N}}`
`\set` 129 `\providecommand\ZZ{\pcsetstyle{Z}}`
`\sequence` 130 `\providecommand\CC{\pcsetstyle{C}}`
`\bin` 131 `\providecommand\QQ{\pcsetstyle{Q}}`
132 `\providecommand\RR{\pcsetstyle{R}}`
133 `\providecommand\PP{\pcsetstyle{P}}`
134 `\providecommand\FF{\pcsetstyle{F}}`
135 `\providecommand\GG{\pcsetstyle{G}}`
136
137 `\providecommand{\set}[1]{\ensuremath{\pc@clrk*{#1}}}`
138 `\providecommand{\sequence}[1]{\ensuremath{\pc@olrk*{#1}}}`
139 `\providecommand{\bin}{\ensuremath{\{0,1\}}}`
140 }

9.1.6 noamsfonts

`\@pc@opt@amsfontsfalse` Package option *noamsfonts* ensures that ams fonts are not loaded. For this flag
`\@pc@opt@amsfontsfalse` is set to false.

```
141 \DeclareOption{noamsfonts}{
142 \@pc@opt@amsfontsfalse
143 }
```

9.1.7 notions

`\indcpa` The *notion* package option defines various cryptographic security notions. The style to
`\indcca` be can be defined via `\pcnotionstyle`.
`\indccai` 144 `\DeclareOption{notions}{`
`\indccaii` 145 `\providecommand{\indcpa}{\pcnotionstyle{IND\pcmathhyphen{CPA}}}`
`\priv` 146 `\providecommand{\indcca}{\pcnotionstyle{IND\pcmathhyphen{CCA}}}`
`\ind` 147 `\providecommand{\indccai}{\pcnotionstyle{IND\pcmathhyphen{CCA1}}}`
`\indcda` 148 `\providecommand{\indccaii}{\pcnotionstyle{IND\pcmathhyphen{CCA2}}}`
`\prvcda` 149 `\providecommand{\priv}{\pcnotionstyle{PRIV}}`
`\prvrda` 150 `\providecommand{\ind}{\pcnotionstyle{IND}}`
`\kia` 151 `\providecommand{\indcda}{\pcnotionstyle{IND\pcmathhyphen{CDA}}}`
`\kda` 152 `\providecommand{\prvcda}{\pcnotionstyle{PRV\pcmathhyphen{CDA}}}`
`\mle` 153 `\providecommand{\prvrda}{\pcnotionstyle{PRV\pcmathhyphen{CDA}}}`
`\uce` 154 `\providecommand{\kia}{\pcnotionstyle{KIAE}}`
`\eufcma` 155 `\providecommand{\kda}{\pcnotionstyle{KDAE}}`
`\eufko` 156 `\providecommand{\mle}{\pcnotionstyle{MLE}}`
`\eufnacma` 157 `\providecommand{\uce}{\pcnotionstyle{UCE}}`
158
`\seufcma` 159 `\providecommand{\eufcma}{\pcnotionstyle{EUF\pcmathhyphen{CMA}}}`
160 `\providecommand{\eufnacma}{\pcnotionstyle{EUF\pcmathhyphen{naCMA}}}`
161 `\providecommand{\seufcma}{\pcnotionstyle{SUF\pcmathhyphen{CMA}}}`
162
163 `\providecommand{\eufko}{\pcnotionstyle{EUF\pcmathhyphen{KO}}}`

164 }

9.1.8 logic

```

\AND
\OR 165 \DeclareOption{logic}{
\NOR 166 % load centernot needed for notimplies
\NOT 167 \@pc@opt@centernottrue
\NAND 168
\XOR 169 \providecommand{\AND}{\ensuremath{\mathrm{AND}}}
\XNOR 170 \providecommand{\OR}{\ensuremath{\mathrm{OR}}}
\xor 171 \providecommand{\NOR}{\ensuremath{\mathrm{NOR}}}
\false 172 \providecommand{\NOT}{\ensuremath{\mathrm{NOT}}}
\true 173 \providecommand{\NAND}{\ensuremath{\mathrm{NAND}}}
\notimplies 174 \providecommand{\XOR}{\ensuremath{\mathrm{XOR}}}
175 \providecommand{\XNOR}{\ensuremath{\mathrm{XNOR}}}
176 \providecommand{\xor}{\ensuremath{\oplus}}
177 \providecommand{\false}{\mathsf{false}}
178 \providecommand{\true}{\mathsf{true}}
179 \providecommand{\notimplies}{\centernot\implies}
180 }

```

9.1.9 ff (function families)

`\kgen` The *ff* option defines macros for function families. Algorithms are typeset via `\pgen` `\pcalgostyle`.

```

\eval 181 \DeclareOption{ff}{
\invert 182 \providecommand{\kgen}{\pcalgostyle{KGen}}
183 \providecommand{\pgen}{\pcalgostyle{Pgen}}
\il 184 \providecommand{\eval}{\pcalgostyle{Eval}}
\ol 185 \providecommand{\invert}{\pcalgostyle{Inv}}
\kl 186
\nl 187 \providecommand{\il}{\pcalgostyle{il}}
\rl 188 \providecommand{\ol}{\pcalgostyle{ol}}
189 \providecommand{\kl}{\pcalgostyle{kl}}
190 \providecommand{\nl}{\pcalgostyle{nl}}
191 \providecommand{\rl}{\pcalgostyle{rl}}
192 }

```

9.1.10 mm (machine models)

`\pcmachinemodelstyle` The *mm* option defines macros for machine models.

```

\CRKT 193 \DeclareOption{mm}{
\TM 194 \providecommand{\CRKT}{\pcmachinemodelstyle{C}}
\PROG 195 \providecommand{\TM}{\pcmachinemodelstyle{M}}
\uTM 196 \providecommand{\PROG}{\pcmachinemodelstyle{P}}
\uC 197
\uP 198 \providecommand{\uTM}{\pcmachinemodelstyle{UM}}
\csize 199 \providecommand{\uC}{\pcmachinemodelstyle{UC}}
\tmtime 200 \providecommand{\uP}{\pcmachinemodelstyle{UEval}}
201
\ppt 202 \providecommand{\csize}{\pcmachinemodelstyle{size}}
203 \providecommand{\tmtime}{\pcmachinemodelstyle{time}}
204 \providecommand{\ppt}{\pcalgostyle{PPT}}
205 }

```

9.1.11 advantage

The *advantage* option defines an `\advantage` command for typesetting advantage declarations of adversaries.

```
206 \DeclareOption{advantage}{
207 \@pc@opt@advantage>true
208 }
```

9.1.12 primitives

`\prover` The *primitives* package option defines various cryptographic primitives.

```
\verifier 209 \DeclareOption{primitives}{
\ nizk      Zero knowledge
\ hash      210 \providecommand{\prover}{\pcalgostyle{P}}
\ gash      211 \providecommand{\verifier}{\pcalgostyle{V}}
\ fash      212 \providecommand{\nizk}{\pcalgostyle{NIZK}}
\ enc       Hash
\ dec
\ sig       213 \providecommand{\hash}{\pcalgostyle{H}}
\ sign      214 \providecommand{\gash}{\pcalgostyle{G}}
\ verify    215 \providecommand{\fash}{\pcalgostyle{F}}
\ obf       216 \providecommand{\pad}{\pcalgostyle{pad}}
\ i0        Encryption
\ di0       217 \providecommand{\enc}{\pcalgostyle{Enc}}
\ owf       218 \providecommand{\dec}{\pcalgostyle{Dec}}
\ prf       Signatures
\ prp       219 \providecommand{\sig}{\pcalgostyle{Sig}}
\ prg       220 \providecommand{\sign}{\pcalgostyle{Sign}}
\ mac       221 \providecommand{\verify}{\pcalgostyle{Vf}}
\puncture   Obfuscation
\ source    222 \providecommand{\obf}{\pcalgostyle{0}}
\ predictor 223 \providecommand{\i0}{\pcalgostyle{i0}}
\ sam       224 \providecommand{\di0}{\pcalgostyle{di0}}
\ dist      One-wayness
\distinguisher
\ simulator 225 \providecommand{\owf}{\pcalgostyle{OWF}}
\ ext       226 \providecommand{\owp}{\pcalgostyle{OWP}}
\ extractor 227 \providecommand{\tdf}{\pcalgostyle{TF}}
           228 \providecommand{\inv}{\pcalgostyle{Inv}}
           229 \providecommand{\hcf}{\pcalgostyle{HC}}
           Pseudorandomness
           230 \providecommand{\prf}{\pcalgostyle{PRF}}
           231 \providecommand{\prp}{\pcalgostyle{PRP}}
           232 \providecommand{\prg}{\pcalgostyle{PRG}}
           Message authentication code
           233 \providecommand{\mac}{\pcalgostyle{MAC}}
           Puncture
           234 \providecommand{\puncture}{\pcalgostyle{Puncture}}
           Misc
           235 \providecommand{\source}{\pcalgostyle{S}}
           236 \providecommand{\predictor}{\pcalgostyle{P}}
           237 \providecommand{\sam}{\pcalgostyle{Sam}}
```

```

238 \providecommand{\dist}{\pcalgostyle{D}}
239 \providecommand{\distinguisher}{\pcalgostyle{Dist}}
240 \providecommand{\simulator}{\pcalgostyle{Sim}}
241 \providecommand{\ext}{\pcalgostyle{Ext}}
242 \providecommand{\extractor}{\ext}
243 }

```

9.1.13 oracles

`\Oracle` The *oracles* package option defines macros for typesetting oracles.

```

\oracle 244 \DeclareOption{oracles}{
  \ro 245 \providecommand{\Oracle}[1]{\pcalgostyle{O{#1}}}
      246
      247 \def\oracle{\bgroup\oracle@}
      248 \newcommand{\oracle@}[1][]{\ifthenelse{equal{#1}{}}{\oracle@@{0}}{\oracle@@{#1}}}
      249 \def\oracle@@#1{\pcoraclestytle{#1}\egroup}
      250
      251 \providecommand{\ro}{\pcoraclestytle{R0}}
      252 }

```

9.1.14 events

`\event` The *events* package option defines macros for typesetting events (probabilistic). Also
`\nevent` defines `\bad` as a *bad event* often used in game based proofs.

```

\bad 253 \DeclareOption{events}{
\nbad 254 \providecommand{\event}[1]{\ensuremath{\mathsf{#1}}}
      255 \providecommand{\nevent}[1]{\ensuremath{\overline{\event{#1}}}}
      256
      257 \providecommand{\bad}{\ensuremath{\event{bad}}}
      258 \providecommand{\nbad}{\ensuremath{\nevent{bad}}}
      259 }

```

9.1.15 complexity

`\complclass` The *complexity* package option defines various complexity classes. The style can be
`\cocomplclass` adjusted via `\pccomplexitystyle`

```

\npol 260 \DeclareOption{complexity}{
\conpol 261 \providecommand{\complclass}[1]{\pccomplexitystyle{#1}}
  \pol 262 \providecommand{\cocomplclass}[1]{\pccomplexitystyle{co}\pcmathhyphen{\pccomplexitystyle{#1}}}
  \bpp 263
\ppoly 264 \providecommand{\npol}{\pccomplexitystyle{NP}}
  \AM 265 \providecommand{\conpol}{\cocomplclass{NP}}
\coAM 266 \providecommand{\pol}{\pccomplexitystyle{P}}
  \AC 267 \providecommand{\bpp}{\pccomplexitystyle{BPP}}
  \NC 268 \providecommand{\ppoly}{\ensuremath{\pol/\mathrm{poly}}}
      269
  \TC 270 \providecommand{\AM}{\pccomplexitystyle{AM}}
  \PH 271 \providecommand{\coAM}{\cocomplclass{AM}}
\csigma 272
  \cpi 273 \providecommand{\AC}[1]{\ensuremath{\ifthenelse{equal{#1}{}}{\pccomplexitystyle{AC}}{\pccomplexitysty
\cosigma 274 \providecommand{\NC}[1]{\ensuremath{\ifthenelse{equal{#1}{}}{\pccomplexitystyle{NC}}{\pccomplexitysty
  \copi 275 \providecommand{\TC}[1]{\ensuremath{\ifthenelse{equal{#1}{}}{\pccomplexitystyle{TC}}{\pccomplexitysty
      276
      277 \providecommand{\PH}{\pccomplexitystyle{PH}}
      278 \providecommand{\csigma}[1]{\pccomplexitystyle{\Sigma}^p_{#1}}

```

```

279 \providecommand{\cpi}[1]{\pccomplexitystyle{\Pi}^p_{#1}}
280 \providecommand{\cosigma}[1]{\cocomplclass{\Sigma}^p_{#1}}
281 \providecommand{\copi}[1]{\cocomplclass{\Pi}^p_{#1}}
282 }

```

9.1.16 asymptotics

`\negl` The *asymptotics* package option defines “polynomials” `c` (`\cc`), `e` (`\ee`), `k` (`\kk`), `m` (`\mm`), `n` (`\nn`), `p` (`\pp`), and `q` (`\qq`) as well as macros `\negl` and `\poly`.

```

\cc 283 \DeclareOption{asymptotics}{
\ee 284 \providecommand{\negl}[1][\secp]{%
\kk 285 \pcpolynomialstyle{\negl}\ifthenelse{\equal{#1}{}}{\pc@olrk*{#1}}
\mm 286
\nn 287 \providecommand{\poly}[1][\secp]{%
\pp 288 \pcpolynomialstyle{\poly}\ifthenelse{\equal{#1}{}}{\pc@olrk*{#1}}
\qq 289
\rr 290 \def\pp{\bgroup\pp@}
291 \newcommand{\pp@}[1][\ifthenelse{\equal{#1}{}}{\pp@@{p}}{\pp@@{#1}}
292 \def\pp@@#1{\pcpolynomialstyle{#1}\egroup}
293
294
295 \providecommand{\cc}{\pcpolynomialstyle{c}}
296 \providecommand{\ee}{\pcpolynomialstyle{e}}
297 \providecommand{\kk}{\pcpolynomialstyle{k}}
298 \providecommand{\mm}{\pcpolynomialstyle{m}}
299 \providecommand{\nn}{\pcpolynomialstyle{n}}
300 \providecommand{\qq}{\pcpolynomialstyle{q}}
301 \providecommand{\rr}{\pcpolynomialstyle{r}}
302 }

```

9.1.17 keys

`\pk` The *keys* package option defines various “keys” such as a symmetric and general purpose `k` (`\key`) or an asymmetric key pair `pk`, `sk` (`\pk` and `\sk`)

```

\sk 303 \DeclareOption{keys}{
\key 304 \providecommand{\pk}{\pckestyle{pk}}
\hk 305 \providecommand{\vk}{\pckestyle{vk}}
\gk 306 \providecommand{\sk}{\pckestyle{sk}}
\fk 307
\st 308 \def\key{\bgroup\key@}
\state 309 \newcommand{\key@}[1][\ifthenelse{\equal{#1}{}}{\key@@{k}}{\key@@{#1}}
310 \def\key@@#1{\pckestyle{#1}\egroup}
311
312 \providecommand{\hk}{\pckestyle{hk}}
313 \providecommand{\gk}{\pckestyle{gk}}
314 \providecommand{\fk}{\pckestyle{fk}}
315
316 \providecommand{\st}{\pckestyle{st}}
317
318 \def\state{\bgroup\state@}
319 \newcommand{\state@}[1][\ifthenelse{\equal{#1}{}}{\state@@{state}}{\state@@{#1}}
320 \def\state@@#1{\pckestyle{#1}\egroup}
321 }

```

9.1.18 Security parameter

`\SECPAR` The n option defines security parameter macros `\secpa` and `\secpa` using n . See
`\secpa` also “lambda” package option.
`\secpa`

```

322 \DeclareOption{n}{
323 \providecommand{\SECPAR}{\ensuremath{\{N_0\}}}
324 \providecommand{\secpa}{\ensuremath{n}}
325 \providecommand{\secpa}{\ensuremath{1^{\secpa}}}
326 }

```

`\SECPAR` The n option defines security parameter macros `\secpa` and `\secpa` using λ . See
`\secpa` also “n” package option.
`\secpa`

```

327 \DeclareOption{lambda}{
328 \renewcommand{\SECPAR}{\ensuremath{\Lambda}}
329 \renewcommand{\secpa}{\ensuremath{\lambda}}
330 \renewcommand{\secpa}{\ensuremath{1^{\secpa}}}
331 }

```

9.2 Preamble and Option Parsing

Print a warning in case an undefined package option is provided.

```

332 \DeclareOption*{%
333 \PackageError{cryptocode}{Unknown option ‘\CurrentOption’}%
334 }

```

By default, only the n option (security parameter as n and 1^n) is loaded

```

335 \ExecuteOptions{n}

```

We are now ready to process all package options

```

336 \ProcessOptions\relax

```

The cryptocode package depends on various external packages which are loaded next. Note that the *amsfonts* package is optional and can be disabled via the *noamsfonts* package option.

Note that *amsmath* and *mathtools* have been loaded already earlier.

```

337
338 \if@pc@opt@amsfonts
339 \RequirePackage{amsfonts}
340 \fi
341 \if@pc@opt@centernot
342 \RequirePackage{centernot}
343 \fi
344 \RequirePackage{xcolor}
345 \RequirePackage{calc}
346 \RequirePackage{tikz}
347 \usetikzlibrary{positioning,calc}
348 \RequirePackage{ifthen}
349 \RequirePackage{xargs}
350 \RequirePackage{pgf}
351 \RequirePackage{forloop}
352 \RequirePackage{array}
353 \RequirePackage{xparse}
354 \RequirePackage{expl3}
355 \RequirePackage{pbox}
356 \RequirePackage{varwidth}
357 \RequirePackage{suffix}

```

```

358 \RequirePackage{etoolbox}
359 \RequirePackage{environ}
360 \RequirePackage{xkeyval}

```

`\pcadvantagesuperstyle` The *advantage* option defines an `\advantage` command for typesetting advantage declarations of adversaries.

```

\pcadvantagename
\pcadvantagesubstyle
\advantage
361 \if@pc@opt@advantage
362 \providecommand{\pcadvantagesuperstyle}[1]{\mathrm{MakeLowercase{#1}}}
363 \providecommand{\pcadvantagesubstyle}[1]{#1}
364 \providecommand{\pcadvantagename}{\mathsf{Adv}}
365
366 \newcommandx*\advantage[3][3=(\secpar)]{\ensuremath{\pcadvantagename^{\pcadvantagesuperstyle{#1}}_{\pcadvantagesubstyle{#1}}}}
367 \fi

```

9.3 Global Macros

9.3.1 Styles

`\pcalgostyle` Definition of styles for algorithms, sets, complexity classes, polynomials, adversaries, notions, keys, and machine models.

`\pcsetstyle`

```

\pccomplexitystyle 368 \providecommand{\pcalgostyle}[1]{\ensuremath{\mathsf{#1}}}
\pcpolynomialstyle 369 \providecommand{\pcsetstyle}[1]{\ensuremath{\mathbb{#1}}}
\pcadvstyle 370 \providecommand{\pccomplexitystyle}[1]{\ensuremath{\mathsf{#1}}}
\pcnotionstyle 371 \providecommand{\pcpolynomialstyle}[1]{\ensuremath{\mathsf{#1}}}
\pckeystyle 372 \providecommand{\pcadvstyle}[1]{\ensuremath{\mathcal{#1}}}
\pcmachinemodelstyle 373 \providecommand{\pcnotionstyle}[1]{\ensuremath{\mathrm{#1}}}
\pcoraclestyle 374 \providecommand{\pckeystyle}[1]{\ensuremath{\mathsf{\protect\vphantom{p}{#1}}}}
375 \providecommand{\pcmachinemodelstyle}[1]{\ensuremath{\mathsf{#1}}}
376 \providecommand{\pcoraclestyle}[1]{\ensuremath{\mathsf{#1}}}

```

9.3.2 Order of Growth

`\pc@olrk` Define order of growth helper macros. These are optionally defined depending on the loaded package options.

`\pc@olrk*`

```

\pc@elrk 377 \DeclarePairedDelimiter\pc@olrk{()}{ }
\pc@elrk* 378 \DeclarePairedDelimiter\pc@elrk{[]}{[]}
\pc@clrk 379 \DeclarePairedDelimiter\pc@clrk{\{ }\{\}}
\pc@clrk*

```

9.3.3 Spacing

`\pcaboveskip` Control the spacing before (resp. after) pseudocode and stacking blocks both vertically and horizontally.

`\pcbelowskip`

```

\pcbeforeskip 380 \newlength\pcaboveskip
\pcafterskip 381 \setlength\pcaboveskip{\abovedisplayskip}
382
383 \newlength\pcbelowskip
384 \setlength\pcbelowskip{\belowdisplayskip}
385
386 \newlength\pcbeforeskip
387 \newlength\pcafterskip

```

9.3.4 Keywords and Highlighting

`\highlightkeyword` Commands for highlighting primary and secondary keywords. Both commands take an optional first parameter to control spacing

`\highlightaltkeyword`

```

388 \newcommand{\highlightkeyword}[2][\ ]{\ensuremath{\mathbf{#2}}\#1}
389 \newcommand{\highlightaltkeyword}[2][\ ]{\ensuremath{\mathsf{#2}}\#1}

```

\pcglobvar All predefined (highlightable) keywords.

```

\pcnew 390 \newcommand{\pcglobvar}{\highlightkeyword{gbl}}
\pcwhile 391 \newcommand{\pcnew}{\highlightkeyword{new}}
\pcendwhile 392 \newcommand{\pcwhile}{\@pc@increaseindent\highlightkeyword{while}}
\pcdo 393 \newcommand{\pcendwhile}{\@pc@decreaseindent\highlightkeyword{endwhile}}
\pcif 394 \newcommandx*\pcdo{2}[1=\ ,2=\ ]{#1\highlightkeyword{#2}{do}}
\pcunless 395 \newcommandx*\pcif{1}[1=\ ]{\@pc@increaseindent\highlightkeyword{#1}{if}}
\pcelse 396 \newcommandx*\pcunless{1}[1=\ ]{\@pc@increaseindent\highlightkeyword{#1}{unless}}
\pcelseif 397 \newcommandx*\pcelse{1}[1=\ ]{\@pc@tmpdecreaseindent\highlightkeyword{#1}{else}}
\pcfi 398 \newcommandx*\pcelseif{1}[1=\ ]{\@pc@tmpdecreaseindent\highlightkeyword{#1}{else if}}
\pcendif 399 \newcommand{\pcfi}{\@pc@decreaseindent\highlightkeyword{fi}}
\pcendfor 400 \newcommand{\pcendif}{\@pc@decreaseindent\highlightkeyword{endif}}
\pcreturn 401 \newcommand{\pcendfor}{\@pc@decreaseindent\highlightkeyword{endfor}}
\pcin 402 \newcommandx*\pcthen{2}[1=\ ,2=\ ]{#1\highlightkeyword{#2}{then}}
\pcfor 403 \newcommand{\pcreturn}{\highlightkeyword{return}}
\pcrepeat 404 \newcommandx*\pcin{2}[1=\ ,2=\ ]{#1\highlightkeyword{#2}{in}}
\pcrepeatuntil 405 \newcommandx*\pcfor{1}[1=\ ]{\@pc@increaseindent\highlightkeyword{#1}{for}}
\pcforeach 406 \newcommand{\pcrepeat}{[1]{%
407 \@pc@increaseindent\ensuremath{%
408 \highlightkeyword{repeat} #1\ \highlightkeyword{times}%
409 }}
\pcendforeach 410 \newcommand{\pcrepeatuntil}[2]{%
\pcuntil 411 \ensuremath{\highlightkeyword{repeat}\ #1\ \highlightkeyword{until}\ #2}}
\pccontinue 412 \newcommand{\pcforeach}{\@pc@increaseindent\highlightkeyword{foreach}}
\pcfalse 413 \newcommand{\pcendforeach}{\@pc@decreaseindent\highlightkeyword{endforeach}}
\pctrue 414 \newcommand{\pcuntil}{\@pc@decreaseindent\highlightkeyword{until}}
\pcnull 415 \newcommand{\pccontinue}{\highlightkeyword{continue}}
\pcdone 416 \newcommandx*\pcfalse{2}[1=\ ,2=\ ]{\highlightkeyword{#2}{false}}
\pcparse 417 \newcommandx*\pctrue{2}[1=\ ,2=\ ]{\highlightkeyword{#2}{true}}
\pcfail 418 \newcommandx*\pcnull{2}[1=\ ,2=\ ]{\highlightkeyword{#2}{null}}
\pcabort 419 \newcommand{\pcdone}{\highlightkeyword{done}}
\pcassert 420 \newcommand{\pcparse}{\highlightkeyword{parse}}
421 \newcommand{\pcfail}{\highlightkeyword{fail}}
422 \newcommand{\pcabort}{\highlightkeyword{abort}}
423 \newcommand{\pcassert}{\highlightkeyword{assert}}

```

9.3.5 Misc

\pcmathhyphen Definition of a hyphen to be used within math formulas.

```
424 \mathchardef\pcmathhyphen ="2D
```

\pccomment Programming style line comment prefixing the comment with a double slash. An optional first parameter allows to control the spacing before the comment (defaults to 1em).

```

425 \newcommand{\pccomment}[2][1em]{\hspace{#1}{\mbox{/!\!/ } \text{\scriptsize#2}}}
426 \newcommand{\pclinecomment}[2][0em]{\hspace{#1}{\mbox{/!\!/ } \text{\scriptsize#2}}}

```

9.4 Internal Helper Functions

\@expandedsetkeys

```
427 \newcommand\@pc@ifinfloat[2]{\ifnum\@floatpenalty<0\relax#1\else#2\fi}
```

`\@expandedsetkeys` Calls `\setkeys` from the `xkeyval` package but before expands argument number 4. Arguments `{\families}` `{\na}` `{\first set of keys}` `{\keys to be expanded}` `{\final set of keys}`

```

428 \newcommand*\@expandedsetkeys[5]{\expandafter\@expandedsetkeys\expandafter{#4}{#1}{#2}{#3}{#5}}
429 \def\@expandedsetkeys@#1#2#3#4#5{\setkeys{#2}[#3]{#4,#1,#5}}

430 \newenvironment{@pc@withspaces}
431 {\obeyspaces\begin{group}\lccode'\sim'\lowercase{\endgroup\let~}\ }
432 {}

```

`\@pc@settowidthofalign` Commands to measure width of an align (resp. aligned) environment. Takes two arguments a length in which to store the resulting width and the content.

```

433 \newcommand{\@pc@settowidthofalign}[2]{%
434   \setbox\z@=\vbox{\@pseudocodecodesize
435     \begin{flalign*}
436       #2
437     \ifmeasuring@\else\global\let\got@maxcolwd\maxcolumnwidths\fi
438     \end{flalign*}
439   }%
440   \begin{group}
441     \def\or{+}\edef\x{\endgroup#1=\dimexpr\got@maxcolwd\relax}\x
442   }
443 \newcommand{\@pc@settowidthofaligned}[2]{%
444   \settowidth{#1}{\@pseudocodesubcodesize$\begin{aligned}#2\end{aligned}$}}

```

`\@pc@ifdraft` Check for draft mode.

```

445 \def\@pc@ifdraft{\ifdim\overfullrule>\z@
446   \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}

```

`\@pc@executeblindly` Run stuff in an empty box

```

447 \newcommand{\@pc@executeblindly}[1]{%
448   \setbox\z@=\vbox{#1 }}

```

We need to fiddle with the label command to use it in `\pseudocode`. To access the original, we store it in

```

449 \AtBeginDocument{
450 \let\@pc@original@label\ltx@label
451 }

```

`\@pc@globaladdtolength` A helper command to set (resp. add to) the length to a given value globally even when being within a scoped grouping.

```

452 \newcommand*{\@pc@globaladdtolength}[2]{%
453 \addtolength{#1}{#2}%
454 \global#1=#1\relax}
455
456 \newcommand*{\@pc@globalsetlength}[2]{%
457 \setlength{#1}{#2}%
458 \global#1=#1\relax}

```

`@pc@global@pc@cnt` A global counter storing the number of times the pseudocode command was triggered.

```

459 \newcounter{@pc@global@pc@cnt}
460 \newcounter{@pc@global@pc@nestcnt}

```


Fix hyperref package.. gnarl <http://tex.stackexchange.com/questions/130319/incompatibility-between-etoolbox-and-hyperref>

```
461 \providecommand{\pcfixhyperref}{
462 \global\let\textlabel\label
463 \global\let\@pc@original@label\textlabel
464 \global\let\@pc@original@label\relax
465 \global\let\label\relax
466 }
```

Allow to support cleveref package. It wants to be loaded after amsmath which is why it needs to be loaded after cryptocode. To do the necessary label fixes, we need to run these after its label fixes at begin document.

```
467 \providecommand{\pcfixcleveref}{
468 \AtBeginDocument{%
469 \pcfixhyperref%
470 \makeatletter%
471 \crefformat{@pc\linenumber}{line-##2##1##3}%
472 \crefrangeformat{@pc\linenumber}{lines-##3##1##4 to-##5##2##6}%
473 \makeatother%
474 }}
```

9.5 Stacking

In the following we define two stacking environments `pchstack` and `pcvstack` to layout multiple pseudocode blocks.

9.5.1 Manual Spacing

```
475 %
476 \newcommand{\pchspace}[1][1em]{\hspace{#1}}
477 \newcommand{\pcvspace}[1][\baselineskip]{\par\vspace{#1}}
478 %
```

9.5.2 Misc

<code>@pc@stackdepth</code> <code>\@pc@incstackdepth</code> <code>\@pc@decstackdepth</code>	Counter to keep track of nesting level of stacks. <pre>479 \newcounter{@pc@stackdepth} 480 \newcommand{\@pc@incstackdepth}{\addtocounter{@pc@stackdepth}{1}} 481 \newcommand{\@pc@decstackdepth}{\addtocounter{@pc@stackdepth}{-1}}</pre>
---	--

9.5.3 Stacking Options

<code>center</code> <code>\@pc@centerstack</code>	Allows to center the stack. <pre>482 \newcommand{\@pc@centerstack}{false} 483 \define@key{pchstack}{center}[true]{\ifthenelse{\equal{#1}{true}} 484 {\renewcommand{\@pc@centerstack}{true}} 485 {\renewcommand{\@pc@centerstack}{false}}}%</pre>
<code>boxed</code> <code>\@pc@boxedstack</code>	Allows to draw a box around the stack. <pre>486 \newcommand{\@pc@boxedstack}{false} 487 \define@key{pchstack}{boxed}[true]{\ifthenelse{\equal{#1}{true}} 488 {\renewcommand{\@pc@boxedstack}{true}} 489 {\renewcommand{\@pc@boxedstack}{false}}}%</pre>

noindent	Allows to draw a box around the stack.
<code>\@pc@noindentstack</code>	<pre> 490 \newcommand{\@pc@noindentstack}{false} 491 \define@key{pcstack}{noindent}[true]{\ifthenelse{\equal{#1}{true}} 492 {\renewcommand{\@pc@noindentstack}{true}} 493 {\renewcommand{\@pc@noindentstack}{false}}}% </pre>
inline	Allows to keep the pchstack inline and not creating a paragraph.
<code>\@pc@inlinestack</code>	<pre> 494 \newcommand{\@pc@inlinestack}{false} 495 \define@key{pcstack}{inline}[true]{\ifthenelse{\equal{#1}{true}} 496 {\renewcommand{\@pc@inlinestack}{true}} 497 {\renewcommand{\@pc@inlinestack}{false}}}% </pre>
space	Introduces horizontal (resp. vertical) space in-between pseudocode blocks in stacking environments .
<code>\pchstackspace</code> <code>\pcvstackspace</code> <code>\@pc@centerstack</code>	<pre> 498 \providecommand{\pchstackspace}{0pt} 499 \providecommand{\pcvstackspace}{0pt} 500 \newcommand{\@pc@stackspace@forpseudocode}{} 501 \newlength{\@pc@stackspace@len} 502 \newcommand*{\@pc@stackspace}{0pt} 503 \newcommand*{\@pc@reset@stackspace}{\setlength{\@pc@stackspace@len}{\@pc@stackspace}} 504 \define@key{pcstack}{space}[0pt]{\renewcommand*{\@pc@stackspace}{#1}}% </pre>
aboveskip	By default <code>\pcaboveskip</code> is applied on the outer most stacking environment. Can be
<code>\@pc@applyaboveskipinstack</code> <code>\overspaceunlessstacking</code>	<p>overridden using <code>aboveskip</code>.</p> <pre> 505 \newcommand{\@pc@addabovespaceunlessstacking}{% 506 \ifthenelse{\value{@pc@stackdepth}=0}{\par\addvspace{\pcaboveskip}}{}} 507 508 \newcommand{\@pc@applyaboveskipinstack}{\@pc@addabovespaceunlessstacking} 509 \let\org@pc@applyaboveskipinstack\@pc@applyaboveskipinstack 510 511 \define@key{pcstack}{aboveskip}[default]{\ifthenelse{\equal{#1}{default}} 512 {\renewcommand{\@pc@applyaboveskipinstack}{\org@pc@applyaboveskipinstack}} 513 {\renewcommand{\@pc@applyaboveskipinstack}{\vspace{#1}}}}% </pre>
belowskip	By default <code>\pcbelowskip</code> is applied on the outer most stacking environment. Can be
<code>\@pc@applybelowskipinstack</code> <code>\lowerspaceunlessstacking</code>	<p>overridden using <code>belowskip</code>.</p> <pre> 514 \newcommand{\@pc@addbelowspaceunlessstacking}{% 515 \ifthenelse{\value{@pc@stackdepth}=0} 516 {\@pc@ifinfloat}{\par\addvspace{\pcbelowskip}}} 517 {} 518 519 \newcommand{\@pc@applybelowskipinstack}{\@pc@addbelowspaceunlessstacking} 520 \let\org@pc@applybelowskipinstack\@pc@addbelowspaceunlessstacking 521 522 \define@key{pcstack}{belowskip}[default]{\ifthenelse{\equal{#1}{default}} 523 {\renewcommand{\@pc@applybelowskipinstack}{\org@pc@applybelowskipinstack}} 524 {\renewcommand{\@pc@applybelowskipinstack}{\par\addvspace{#1}}}}% </pre>
<code>\pcbeforehstackskip</code> <code>\pcafterhstackskip</code>	<p>Allows adding global skips before and after <code>\pchstack</code> blocks.</p> <pre> 525 \newlength{\pcbeforehstackskip} 526 \newlength{\pcafterhstackskip} </pre>
<code>\@pc@boxedstack</code>	<p>For <code>\pchstack</code> and <code>\pcvstack</code> we use a box to store temporary results.</p> <pre> 527 \newsavebox{\@pc@stackcontentbox}% </pre>

```

\pcsethstackargs
\pcsetvstackargs 528 \newcommand*\@pc@hstack@defaultargs{}
529 \newcommand*\pcsethstackargs[1]{\renewcommand*\@pc@hstack@defaultargs{#1}}
530 \newcommand*\@pc@vstack@defaultargs{}
531 \newcommand*\pcsetvstackargs[1]{\renewcommand*\@pc@vstack@defaultargs{#1}}

```

9.5.4 The Stacking Environments

pccenter

```

532 \newenvironment{pccenter}{%
533 \setlength\topsep{0pt}\setlength\parskip{0pt}%
534 \begin{center}}{\end{center}}

```

pchstack A stacking environment for horizontally stacked pseudocode blocks.

```

535 \NewEnviron{pchstack}[1][]{%
536 %Ensure that the parameters are defaulted
537 \begingroup%
538 % parse args this is the same as
539 % \setkeys{pcstack}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pchstackspace,%
540 % expect that we expand the default args
541 \@expandedsetkeys{pcstack}{}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pchstackspace,%
542 \@pc@reset@stackspace%
543 %add above skip except when in inline mode
544 \ifthenelse{\equal{\@pc@inlinestack}{true}}{}{\@pc@applyaboveskipinstack}%
545 \@pc@incstackdepth%
546 \renewcommand{\@pc@stackspace@forpseudocode}{\hspace{\@pc@stackspace}}%
547 %Store main content in a box
548 \ifthenelse{\equal{\@pc@boxedstack}{true}}{%
549 {\sbox{\@pc@stackcontentbox}
550   {\fbox{\mbox{\hspace{\pcbeforehstackskip}\BODY\hspace{\pcafterhstackskip}\hspace{-\@pc@stackspace}}}}%
551 {\sbox{\@pc@stackcontentbox}
552   {\mbox{\hspace{\pcbeforehstackskip}\BODY\hspace{\pcafterhstackskip}\hspace{-\@pc@stackspace}}}}}%
553 % handle noindent
554 \ifthenelse{\equal{\@pc@noindentstack}{true}}{\par\noindent\ignorespaces}{}%
555 %set content either centered or directly
556 \ifthenelse{\equal{\@pc@centerstack}{true}}{%
557 {\begin{pccenter}\usebox{\@pc@stackcontentbox}\end{pccenter}}
558 {\usebox{\@pc@stackcontentbox}}}%
559 % cleanup
560 \@pc@decstackdepth%
561 \ifthenelse{\equal{\@pc@inlinestack}{true}}{}{\@pc@applybelowskipinstack}%
562 \endgroup%reset space outside group
563 \@pc@reset@stackspace%
564 \@pc@stackspace@forpseudocode%
565 %ignore any spaces after, to allow staying within paragraph
566 \ignorespacesafterend\noindent%
567 }

```

pchstack A stacking environment for vertically stacked pseudocode blocks.

```

568 \NewEnviron{pcvstack}[1][]{%
569 %Ensure that the parameters are defaulted
570 \begingroup%
571 % parse args this is the same as
572 % \setkeys{pcstack}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pcvstackspace,%
573 % expect that we expand the default args

```

```

574 \@expandedsetkeys{pcstack}{-}{center=false,boxed=false,aboveskip=default,belowskip=default,space=\pcvst
575 \@pc@reset@stackspace%
576 \@pc@applyaboveskipinstack%
577 \@pc@incstackdepth%
578 \renewcommand{\@pc@stackspace@forpseudocode}{\par\vspace{\@pc@stackspace}}%
579 %Store main content in a box
580 \sbox{\@pc@stackcontentbox}{%
581 \ifthenelse{equal{\@pc@boxedstack}{true}}%
582 {\fbox{\raisebox{\dimexpr\ht\strutbox-\height}{\begin{varwidth}[t]{2\linewidth}\BODY\end{varwidth}}}}%
583 {\raisebox{\dimexpr\ht\strutbox-\height}{\begin{varwidth}[t]{2\linewidth}\BODY\end{varwidth}}}%
584 \vspace{-\@pc@stackspace}}%
585 % handle noindent
586 \ifthenelse{equal{\@pc@noindentstack}{true}}{\par\noindent\ignorespaces}{}%
587 % display content
588 \ifthenelse{equal{\@pc@centerstack}{true}}%
589 {\begin{pccenter}\usebox{\@pc@stackcontentbox}\end{pccenter}}%
590 {\usebox{\@pc@stackcontentbox}}%
591 % cleanup
592 \@pc@decstackdepth%
593 \@pc@applybelowskipinstack%
594 \endgroup%reset space outside group
595 \@pc@reset@stackspace%
596 \@pc@stackspace@forpseudocode%
597 %ignore any spaces after, to allow staying within paragraph
598 \ignorespacesafterend\noindent%
599 }

```

9.6 The pseudocode command

Define internal lengths used for measurements within pseudocode.

```

600 \newlength{\@pc@minipage@length}
601 \newlength{\@pc@alt@minipage@length}
602 \newlength{\@pc@length@tmp@width@vstack}

```

Define flags used in game based proofs.

```

603 \newcommand{\@withingame}{false}
604 \newcommand{\@withinbxgame}{false}
605 \newcommand{\@withingamedescription}{false}

```

`\@bxgameheader` Define a placeholder command which will take the current game header.

```

606 \newcommand{\@bxgameheader}{}

```

`\@pc@beginnewline` An internal helper that is called at the beginning of each new line.

```

607 \newlength\@pseudocodecodeminilineheight@len
608 \newcommand{\@pc@beginnewline}{%
609 \@pseudocodecodeatbeginline\@pseudocodelinenummer\@pc@and\@pc@ln@stephiddenlncnt%
610 \setlength{\@pseudocodecodeminilineheight@len}{\@pseudocodecodeminilineheight}%
611 \vphantom{\rule[0.5ex-0.5\@pseudocodecodeminilineheight@len]{0pt}{\@pseudocodecodeminilineheight@len}}%
612 %checkspace
613 \ifthenelse{equal{\@pseudocodespace}{auto}}%
614 {\expandafter\pcind\expandafter[\value{\@pc@indentationlevel}]}%
615 }%
616 %reset column counter
617 \setcounter{pccolumncounter}{2}%
618 %beginmode
619 \@pc@modebegin}

```

`\@pc@and@wrap@end` Every pseudocode line is wrapped in between `\@pc@and@wrap@start` and `\@pc@and@wrap@end`.
`\@pc@and@wrap@start` 620 `\newcommand{\@pc@and@wrap@start}{\@pc@beginnewline}`
621 `\newcommand{\@pc@and@wrap@end}{\@pc@modeend&\@pseudocodecodeatendline}`

`\@pc@and` An internal helper to store the ampersand. As this is a special character this is the easiest in order to place custom alignment tags.
622 `\newcommand{\@pc@and}{&}`

`\pcind` An indentation macro to be used within pseudocode. As writing `\pcind` is a bit cumbersome, there is a shorthand that can be defined via `\pcindentname` (defaults to t). See below.
623 `\newlength{\@pcindentwidth}`
624 `\providecommand{\pcind}[1][1]{%`
625 `\setlength{\@pcindentwidth}{\widthof{\ensuremath{\quad}}*#1}%`
626 `\ensuremath{\mathmakebox[\@pcindentwidth]{}}`

`\pctabname` Shorthands for alignment tabs and indentation. These are defined only within the pseudocode scope.
`\pcdbltabname`
`\pcindentname` 627 `\newcommand{\pctabname}{>}`
628 `\newcommand{\pcdbltabname}{<}`
629 `\newcommand{\pcindentname}{t}`

The following commands handle line numbering within the pseudocode command. The pseudocode command itself does need to do some counter magic. We start with a definition of various helper counters. The H version of counters is needed to make hyperref happy

630 `\newcounter{pclinenum}`
631 `\newcounter{Hpclinenum}`
632 `\newcounter{@pclinenum}`
633 `\newcounter{Hpclinenum}`
634 `\newcounter{@pclinenumbertmp}`
635 `\newcounter{pcgamecounter}`
636 `\newcounter{Hpgamecounter}`
637 `\newcounter{pcrlinenum}`
638 `\newcounter{Hpcrlinenum}`
639 `\newcounter{@pcrlinenumbertmp}`

The following implements some counter magic. When using automatic line numbering line numbers are nicely aligned before the first alignment tag. This, however confuses hyperref and we thus have a second counter that is updated after the first tag. This is done with the `\@pcln@stephiddenlncnt`

640 `\renewcommand{\the@pclinenum}{\thepclinenum}`
641 `\providecommand{\@pcln@stephiddenlncnt}{%`
642 `\refstepcounter{@pclinenum}%`
643 `\stepcounter{Hpclinenum}%`
644 `}`

`\pclnseparator` Define separators between line numbers and code (left and right). Note that line numbers
`\pcrlnseparator` can be displayed either to the left or to the right of code.
645 `\providecommand{\pclnseparator}{:}`
646 `\providecommand{\pcrlnseparator}{}`

`\pclnspace` Define spacing between line numbers and code (left and right).
`\pclnspace` 647 `\providecommand{\pclnspace}{1em}`
648 `\providecommand{\pclnspace}{0.5em}`

`\pclnstyle`

```

649 \providecommand\pclnstyle[1]{\text{\scriptsize#1}}

pcln Manually place (left aligned) line numbers. This command is also used by the automatic
placement of line numbers.
650 \providecommand{\pcln}{%
651 \ifthenelse{\equal{\@pc@skiplnmarker}{1}}{\ifmeasuring@else\@pc@resetskipln{\fi}}{%
652 \refstepcounter{pclinenum}}%
653 \stepcounter{Hpclinenum}}%
654 \ifthenelse{\value{pclinenum}<10}{\hspace{1ex}}{}%
655 \pclnstyle{\arabic{pclinenum}}\pclnseparator\hspace{\pclnsp}}%
656 }%

\pcskipln allow to skip numbering single lines if linenumbering=on
\@pc@skiplnmarker 657 \def\@pc@skiplnmarker{}
skipfirstln 658 \providecommand{\pcskipln}{\ifmeasuring@else\global\def\@pc@skiplnmarker{1}\fi}
659 \newcommand{\@pc@resetskipln}{\global\def\@pc@skiplnmarker{}}
660 \define@key{pseudocode}{skipfirstln}[1]{\global\def\@pc@skiplnmarker{1}}

\pclnr Manual placement of right aligned line numbers using the same counter (\pclnr) or a
\pcrln separate counter (\pcrln).
661 \providecommand{\pclnr}{%
662 \refstepcounter{pclinenum}}%
663 \stepcounter{Hpclinenum}}%
664 \hspace{\pclnrsp}\pclnseparator\pclnstyle{\arabic{pclinenum}}%
665
666 \providecommand{\pcrln}{%
667 \refstepcounter{pcrlinenum}}%
668 \stepcounter{Hpcrlinenum}}%
669 \hspace{\pcrlrsp}\pcrlseparator\pclnstyle{\arabic{pcrlinenum}}%

```

9.6.1 Options

The following commands define a bunch of placeholders (plus their default values) that are defined via the various options of the pseudocode command.

```

670 \newcommand*\@pseudocodehead{}
671 \newcommand*\@pseudocodewidth{}
672 \newcommand*\@pseudocodexshift{0pt}
673 \newcommand*\@pseudocodeyshift{0pt}
674 \newcommand*\@pseudocodelinenum{}
675 \newcommand*\@pseudocodebeforeskip{0ex}
676 \newcommand*\@pseudocodeafterskip{0ex}
677 \newcommand*\@pseudocodelnstart{0}
678 \newcommand*\@pseudocodelnstartright{0}
679 \newcommand*\@pseudocodesyntaxhighlighting{}
680 \newcommand*\@pseudocodenodraft{false}
681 \newcommand*\@pseudocodecolspace{} % empty per default, use length,
682
683 \newcommand*\@pseudocodeheadlinecmd{\hrule}
684

headlinesep Distance between header and line.
\pcheadlinesep 685 \newlength\pcheadlinesep
@pseudocodeheadlinesep 686 \setlength\pcheadlinesep{0pt}
687 \newcommand*\@pseudocodeheadlinesep{0em}
688 \define@key{pseudocode}{headlinesep}[0em]{\renewcommand*\@pseudocodeheadlinesep{#1}}

```

```

bodylinesep
\pcbodylinesep 689 \newlength\pcbodylinesep
\pseudocodebodylinesep 690 \setlength\pcbodylinesep{0.3\baselineskip}
691 \newcommand*\@pseudocodebodylinesep{0em}
692 \define@key{pseudocode}{bodylinesep}[0em]{\renewcommand*\@pseudocodebodylinesep{#1}}

headheight
\pcheadheight 693 \newlength\@pseudocodeheadheight@len
\@pc@headheightskip 694 \newcommand{\@pc@headheightskip}{%
\pseudocodeheadheight 695 \setlength{\@pseudocodeheadheight@len}{\@pseudocodeheadheight}%
696 \vphantom{\rule[0.5ex-0.5\@pseudocodeheadheight@len]{0pt}{\@pseudocodeheadheight@len}}}%
697 }
698 \newlength\pcheadheight
699 \setlength{\pcheadheight}{3.25ex}
700 \newcommand*\@pseudocodeheadheight{\pcheadheight}
701 \define@key{pseudocode}{headheight}[0em]{\renewcommand*\@pseudocodeheadheight{#1}}

702
703 \newcommand*\@pseudocodecolsep{0em}
704 \newcommand*\@pseudocodeaddtolength{2pt}
705
706 \newcommand*\@pseudocodecodeatbeginline{}
707 \newcommand*\@pseudocodecodeatendline{}
708 \newcommand*\@pseudocodecodejot{0em}
709 \newcommand*\@pseudocodecodesize{\small}
710 \newcommand*\@pseudocodesubcodesize{\footnotesize}
711
712 \newcommand*\@pseudocodeminipagealign{t}
713
714 %
715 % Define keywords for the automatic syntax highlighting
716 % the accompanying add provides additional keywords.
717 % The space version for automatic spacing
718 \newcommand*\@pseudocodekeywordsindent{for ,foreach ,if ,repeat ,while }
719 \newcommand*\@pseudocodekeywordsunindent{endfor,endforeach,fi,endif,until,endwhile}
720 \newcommand*\@pseudocodekeywordsuninindent{elseif ,else }
721 \newcommand*\@pseudocodekeywords{for,foreach,{return },return,{ do },{ in },new,if , null , true,{until }
722 \newcommand*\@pseudocodeaddkeywords{}
723 \newcommand*\@pseudocodealtkeywords{}
724 \begin{@pc@withspaces}
725 \global\def\@pseudocodekeywordsspace{for,endfor,foreach,endforeach,return,do,in,new,if,null,true,until
726 \end{@pc@withspaces}

```

Specification of the various options of the \pseudocode command.

```

727 \define@key{pseudocode}{beginline}[]{\renewcommand*\@pseudocodecodeatbeginline{#1}}
728 \define@key{pseudocode}{endline}[]{\renewcommand*\@pseudocodecodeatendline{#1}}
729 \define@key{pseudocode}{jot}[0em]{\renewcommand*\@pseudocodecodejot{#1}}
730 \define@key{pseudocode}{codesize}[\small]{\renewcommand*\@pseudocodecodesize{#1}}
731 \define@key{pseudocode}{subcodesize}[\small]{\renewcommand*\@pseudocodesubcodesize{#1}}
732 \define@key{pseudocode}{head}[]{\renewcommand*\@pseudocodehead{#1}}
733 \define@key{pseudocode}{width}[]{\renewcommand*\@pseudocodewidth{#1}}
734 \define@key{pseudocode}{valign}[t]{\renewcommand*\@pseudocodeminipagealign{#1}}
735 \define@key{pseudocode}{xshift}[]{\renewcommand*\@pseudocodexshift{#1}}
736 \define@key{pseudocode}{yshift}[]{\renewcommand*\@pseudocodeyshift{#1}}
737 \define@key{pseudocode}{colspace}[]{\renewcommand*\@pseudocodecolspace{#1}}
738 \define@key{pseudocode}{linenumbering}[on]{\ifthenelse{equal{#1}{on}}{\renewcommand*\@pseudocodelinen

```

```

739 \define@key{pseudocode}{before skip}[]{\renewcommand*\@pseudocodebefore skip{#1}}
740 \define@key{pseudocode}{after skip}[]{\renewcommand*\@pseudocodeafter skip{#1}}
741 \define@key{pseudocode}{l nstart}[0]{\renewcommand*\@pseudocodel nstart{#1}}
742 \define@key{pseudocode}{l nstart right}[0]{\renewcommand*\@pseudocodel nstart right{#1}}
743 \define@key{pseudocode}{col sep}[0em]{\renewcommand*\@pseudocodecol sep{#1}}
744 \define@key{pseudocode}{headline cmd}[\hrule]{\renewcommand*\@pseudocodeheadline cmd{#1}}
745 \define@key{pseudocode}{add to length}[2pt]{\renewcommand*\@pseudocodeadd to length{#1}}
746 \define@key{pseudocode}{no draft}[true]{\renewcommand*\@pseudocodenodraft{#1}}
747 \define@key{pseudocode}{keywords}[]{\renewcommand*\@pseudocodekeywords{#1}}
748 \define@key{pseudocode}{keywords indent}[]{\renewcommand*\@pseudocodekeywords indent{#1}}
749 \define@key{pseudocode}{keywords unindent}[]{\renewcommand*\@pseudocodekeywords unindent{#1}}
750 \define@key{pseudocode}{keywords uninindent}[]{\renewcommand*\@pseudocodekeywords uninindent{#1}}
751 \define@key{pseudocode}{add keywords}[]{\renewcommand*\@pseudocodeadd keywords{#1}}
752 \define@key{pseudocode}{alt keywords}[]{\renewcommand*\@pseudocodealt keywords{#1}}
753 \define@key{pseudocode}{syntax highlight}[]{\renewcommand*\@pseudocodesyntax highlighting{#1}}

```

mode The [*mode*] key (with values *text* or *math* (default)) specifies whether within a pseudocode block input is by default typeset in text mode or in math mode. The \@pc... variables are variables that help typesetting each line in a pseudocode block.

```

754 \newcommand{\@pc@mode begin}{ }
755 \newcommand{\@pc@mode end}{ }
756 \define@key{pseudocode}{mode}[math]{ %
757 \ifthenelse{\equal{#1}{text}}{ %
758 \renewcommand*\@pc@mode begin{\begin{varwidth}{\textwidth}} %
759 %introduce line magic for text mode
760 \let\@pc@lb\ }
761 \renewcommandx*{\ }[2][1=,2=]{\@pc@mode end\@pc@and \@pseudocodecode at end line\ifthenelse{\equal{###1}{ }
762 \def\pclb{\let\@pc@lb\relax\@pc@mode end\ } %
763 \def\pcolb{\let\@pc@lb\relax\@pc@mode end\ } %
764 } %
765 \renewcommand*\@pc@mode end{\end{varwidth}} %
766 {\renewcommand{\@pc@mode begin}{ }\renewcommand{\@pc@mode end}{ } }

```

minlineheight Control the minimal line height of pseudocode blocks.

```

\pcminlineheight 767 \providecommand{\pcminlineheight}{0pt}
codecodeminlineheight 768 \newcommand*\@pseudocodecodeminlineheight{\pcminlineheight}
769 \define@key{pseudocode}{minlineheight}[0pt]{\renewcommand*\@pseudocodecodeminlineheight{#1}}

```

9.6.2 Automatic Syntax Highlighting and Spacing (Experimental)

Experimental LaTeX3 string substitution helpers for automatic keyword highlighting. The regex parsing is (regrettably) super slow.

```

770 \ExplSyntaxOn
771 \tl_new:N \l_pc_strsub_input_tl
772 \tl_new:N \l_pc_strsub_search_tl
773 \tl_new:N \l_pc_strsub_replace_tl
774
775 \NewDocumentCommand{\@pc@stringsubstitution}{mmm}
776 {
777 \tl_set:Nn \l_pc_strsub_input_tl { #1 }
778 \tl_set:Nn \l_pc_strsub_search_tl { #2 }
779 \tl_set:Nn \l_pc_strsub_replace_tl { #3 }
780 % \tl_show_analysis:N \l_pc_strsub_input_tl % uncomment for debugging
781 % \tl_show_analysis:N \l_pc_strsub_search_tl % uncomment for debugging
782 % \tl_show_analysis:N \l_pc_strsub_replace_tl % uncomment for debugging

```



```

783 \regex_replace_all:nnN
784 { \u{l_pc_strsub_search_tl} } %only match if keyword does not have a word character preceding
785 { \u{l_pc_strsub_replace_tl} }
786 \l_pc_strsub_input_tl
787 % \tl_show_analysis:N \l_tmpa_tl % uncomment for debugging
788 \tl_use:N \l_pc_strsub_input_tl
789 }
790 \ExplSyntaxOff

```

\@pc@syntaxhighlight
\@pc@highlight
\@pc@highlightindent
\@pc@highlightunindent
\@pc@highlightunindent
\@pc@alhighlight

This is the core of the (experimental) automatic syntax highlighting and automatic spacing. The code is ugly, and very slow. It is not really recommended to be used in larger projects.

```

791 \newcommand{\@pc@syntaxhighlight}[1]{%
792 %don't highlight during measuring runs for performance improvements.
793 \ifmeasuring@#1\else%
794 \ifthenelse{\equal{\@pseudocodesyntaxhighlighting}{auto}}{%
795 \def\@shtmp{#1}% first step
796 % Depending on space mode, we might later run the indent/unindent/... lists
797 % if not, we add them now to tmp lists in order to have a complete list.
798 \ifthenelse{\equal{\@pseudocodespace}{keep}}
799   {\edef\@tmpkeywords{\@pseudocodekeywordsspace,\@pseudocodeaddkeywords}}
800   {\ifthenelse{\equal{\@pseudocodespace}{auto}}
801     {\edef\@tmpkeywords{\@pseudocodekeywords,\@pseudocodeaddkeywords}}
802     {\edef\@tmpkeywords{\@pseudocodekeywords,\@pseudocodekeywordsindent,\@pseudocodekeywordssunindent}}
803 \foreach \@pckw in \@tmpkeywords{%
804 \ifthenelse{\equal{\@pckw}{}}{ }{%
805 % we are doing a simple strsub and storing the result (globally) in \@shtmp
806 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
807   \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
808   \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
809   {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
810   \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
811   {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
812   }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
813 }}}% alt keywords
814 \foreach \@pckw in \@pseudocodealtkeywords{%
815 \ifthenelse{\equal{\@pckw}{}}{ }{%
816 % we are doing a simple strsub and storing the result (globally) in \@shtmp
817 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
818   \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
819   \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
820   {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
821   \@pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
822   {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
823   }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
824 }}}%
825 % if automatic spacing
826 \ifthenelse{\equal{\@pseudocodespace}{auto}}
827 {%
828 \foreach \@pckw in \@pseudocodekeywordsindent{% indentation keywords
829 \ifthenelse{\equal{\@pckw}{}}{ }{%
830 % we are doing a simple strsub and storing the result (globally) in \@shtmp
831 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
832   \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
833   \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
834   {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter

```

```

835 \pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\exp
836 {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
837 }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
838 }}%
839 \foreach \@pckw in \@pseudocodekeywords\unindent{% unindentation keywords
840 \ifthenelse{\equal{\@pckw}{}}{\}%
841 % we are doing a simple strsub and storing the result (globally) in @shtmp
842 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
843 \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
844 \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
845 {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
846 \pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\exp
847 {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
848 }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
849 }}%
850 \foreach \@pckw in \@pseudocodekeywords\uninindent{% uninindentation keywords
851 \ifthenelse{\equal{\@pckw}{}}{\}%
852 % we are doing a simple strsub and storing the result (globally) in @shtmp
853 \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
854 \gdef\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
855 \@shtmp\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
856 {\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
857 \pc@stringsubstitution\expandafter\expandafter\expandafter\expandafter\expandafter\exp
858 {\expandafter\expandafter\expandafter\@shtmp\expandafter\expandafter\expandafter
859 }\expandafter\expandafter\expandafter{\expandafter\@pckw\expandafter}\expandafter{\exp
860 }}%
861 \}%
862 % return result
863 \@shtmp%
864 \ifthenelse{\equal{\@pckw}{}}{\}%
865 \fi}
866
867 \newcommand{\@pc@highlight}[1]{%
868 \ifthenelse{\equal{\@pseudocodespace}{keep}}
869 {\highlightkeyword[] {#1}}%
870 {\highlightkeyword[] {\@pc@stringsubstitution{#1}{ }{-}}}%
871 }
872
873 \newcommand{\@pc@highlightindent}[1]{%
874 \@pc@increaseindent\@pc@highlight{#1}%
875 }
876
877 \newcommand{\@pc@highlightunindent}[1]{%
878 \@pc@decreaseindent\@pc@highlight{#1}%
879 }
880
881 \newcommand{\@pc@highlightuninindent}[1]{%
882 \@pc@tmpdecreaseindent\@pc@highlight{#1}%
883 }
884
885 \newcommand{\@pc@althighlight}[1]{%
886 \ifthenelse{\equal{\@pseudocodespace}{keep}}
887 {\highlightaltkeyword{#1}}%
888 {\highlightaltkeyword{\@pc@stringsubstitution{#1}{ }{-}}}%
889 }

```

9.6.3 Helper Variables

```

\@pc@thecontent Helper variables used within pseudocode
\@pc@colspace 890 \newcommand{\@pc@thecontent}{}
891 \newcommand{\@pc@colspace}{}

\@withinspaces Helper variables for controlling automatic spacing
\@keepspaces 892 \newcommand{\@withinspaces}{false}%
893 \newcommand{\@keepspaces}{%
894 \renewcommand{\@withinspaces}{true}\@pc@withspaces%
895 }

896 \newcommand*\@pseudocodespace{}
897 \define@key{pcspace}{space}[]{\ifthenelse{equal{#1}{keep}}{\@keepspaces}{}\renewcommand*\@pseudocodespace{#1}}
898
899
900 \newcommand*\@pc@defaultargs{}
901 \newcommand*\pcsetargs[1]{\renewcommand*\@pc@defaultargs{#1}}
902
903 % automatic indentation
904 \newcounter{\@pc@indentationlevel}
905 \newcommand{\@pc@increaseindent}{\addtocounter{\@pc@indentationlevel}{1}}
906 \newcommand{\@pc@decreaseindent}{\ifthenelse{equal{\@pseudocodespace}{auto}}{\pcind[-1]}{\addtocounter{\@pc@indentationlevel}{-1}}}
907 \newcommand{\@pc@tmpdecreaseindent}{\ifthenelse{equal{\@pseudocodespace}{auto}}{\pcind[-1]}{}}
908
909 \newcounter{pccolumncounter}
910 \setcounter{pccolumncounter}{2}
911
912 % store original halign
913 \let\@pc@halign\halign%
```

9.6.4 The Actual Pseudocode Command

```

914 % Check if the pseudocode command is called with an optional argument
915 \providecommand{\pseudocode}{%
916 \begingroup%
917 \renewcommand{\@withinspaces}{false}%
918 \@ifnextchar [%]
919 {\@pseudocodeA}%
920 {\@pseudocode[]}%
921 }
922
923 \def\@pseudocodeA[#1]{%
924 \setkeys{pcspace}{#1}%test if there is a space assignment within the keys .. make the necessary arrangements
925 \@pseudocode[#1]%
926 }
927
928 \def\@pseudocode[#1]#2{%
929 \begingroup%
930 % reset skip marker before parsing options, as this might set it
931 \@pc@resetskipln%
932 % parse options
933 % this is the same as %\setkeys{pseudocode}[space]{\@pc@defaultargs,#1}%ignore the space key.
934 % expect that we expand the default args
935 \@expandedsetkeys{pseudocode}[space]{head=}{\@pc@defaultargs}{#1}%
936 % check draft mode and disable syntax highlighting
```

```

937 \@pc@ifdraft{\ifthenelse{\equal{\@pseudocodenodraft}{true}}{\renewcommand\@pseudocodesyntaxhighlight%
938 %
939 %
940 \addtocounter{\@pc@global@pc@nestcnt}{1}%
941 % allow for tikz usage
942 \@pc@ensureremember%
943 %
944 % create tabbing command
945 \ifcsname \pctabname\endcsname%
946 \expandafter\renewcommand\csname \pctabname\endcsname{\@pc@modeend&\@pc@colspace\@pc@modebegin}%
947 \else%
948 \expandafter\newcommand\csname \pctabname\endcsname{\@pc@modeend&\@pc@colspace\@pc@modebegin}%
949 \fi%
950 \ifcsname \pcdbltabname\endcsname%
951 \expandafter\renewcommand\csname \pcdbltabname\endcsname{\@pc@modeend&\@pc@colspace\@pc@modebegin}%
952 \else%
953 \expandafter\newcommand\csname \pcdbltabname\endcsname{\@pc@modeend&\@pc@colspace\@pc@modebegin}%
954 \fi%
955 % create colspace command if necessary (must be empty for multicolumns
956 \ifthenelse{\equal{\@pseudocodecolspace}{}}{
957 {}
958 {\renewcommand{\@pc@colspace}{\hspace{\@pseudocodecolspace}}}%
959 %
960 %adjust row width
961 \addtolength{\jot}{\@pseudocodecodejot}%
962 % create indent command
963 \expandafter\let\csname \pcindentname\endcsname\pcind%
964 %
965 %store and wrap (do syntax highlighting) argument
966 \renewcommand{\@pc@thecontent}{\@pc@and@wrap@start\@pc@syntaxhighlight{#2}\@pc@and@wrap@end}%
967 %
968 %take care of counters
969 \stepcounter{\@pc@global@pc@cnt}%
970 \setcounter{\pcclinenumber}{\@pseudocodelnstart}%
971 \setcounter{\pcrlinenumber}{\@pseudocodelnstartright}%
972 \setlength{\@pc@minipage@length}{0pt}%
973 \setlength{\@pc@alt@minipage@length}{0pt}%
974 \setcounter{\pcclinenumbertmp}{\value{\pcclinenumber}}%
975 \setcounter{\pcrlinenumbertmp}{\value{\pcrlinenumber}}%
976 %reset column counter
977 \setcounter{\pccolumncounter}{2}%
978 %
979 % vertical space
980 \vspace{\@pseudocodeyshift}%
981 %
982 %
983 %
984 % line magic
985 \ifthenelse{\value{\@pc@global@pc@nestcnt}=1}{%
986 \let\@pc@halign\halign%
987 \newenvironment{pcmbbox}{\let\halign\@pc@halign}{}%
988 \def\halign{%
989 \renewcommand{\label}[1]{\ifmeasuring@else\@pc@original@label{###1}\fi}%
990 \let\@pc@lb\\%
991 \renewcommandx*{\}[2][1=,2=]{\@pc@modeend\@pc@and\@pseudocodecodeatendline \ifthenelse{\equal{###1}{%
992 \def\pclb{\let\\ \@pc@lb\relax\@pc@modeend\\}%

```

```

993 \@pc@halign}%
994 }{}%
995 %
996 %align column separation
997 \renewcommand*{\minalignsep}{\@pseudocodecolsep}%
998 %
999 %as the following block will execute the pseudocode we need to store the skip command
1000 \edef\@pc@org@skiplnmarker{\@pc@skiplnmarker}%
1001 % if no width is set compute width and store in circuitlength
1002 \ifthenelse{\equal{\@pseudocodewidth}{}}{%
1003 % compute length of pseudocode
1004 \ifthenelse{\value{\@pcsubprogstep}=0}{%
1005 \@pc@settowidthofalign{\@pc@minipage@length}{\@pc@thecontent}%
1006 }{%
1007 \@pc@settowidthofaligned{\@pc@minipage@length}{\@pc@thecontent}%
1008 }%
1009 %compute length of header
1010 \ifthenelse{\equal{\@withingame}{true}}{%
1011 {\ifthenelse{\equal{\@pc@secondheader}{true}}%
1012 {\addtolength{\@pc@alt@minipage@length}{\widthof{x\ensuremath{\@pc@gametitle[1]\@pc@gametitle[1]}}}%
1013 {\addtolength{\@pc@alt@minipage@length}{\widthof{\ensuremath{\@pc@gametitle[1]}}}}}%
1014 {\addtolength{\@pc@alt@minipage@length}{\widthof{\@pseudocodehead}}}%
1015 % use header length if longer and add some points for good measure
1016 \ifdim\@pc@alt@minipage@length>\@pc@minipage@length%
1017 \setlength{\@pc@minipage@length}{\@pc@alt@minipage@length}%
1018 \fi%
1019 \addtolength{\@pc@minipage@length}{\@pseudocodeaddtolength}%
1020 }\addtolength{\@pc@minipage@length}{\@pseudocodewidth}}%
1021 % reset counter and skip command
1022 \setcounter{pclinenum}{\value{\@pcclinenumbertmp}}%
1023 \setcounter{pcrlinenumber}{\value{\@pcrlinenumbertmp}}%
1024 \setcounter{\@pc@indentationlevel}{0}%
1025 \edef\@pc@skiplnmarker{\@pc@org@skiplnmarker}%
1026 % begin actual output
1027 %
1028 %
1029 %do the actual mini page
1030 \hspace{\pcbeforeskip}\hspace{\@pseudocodexshift}%
1031 \ifthenelse{\equal{\@pseudocodeminipagealign}{t}}{%
1032 \raisebox{\dimexpr\ht\strutbox-\height}{\@pc@pseudocodeminipage{t}}%
1033 }{%
1034 \@pc@pseudocodeminipage{\@pseudocodeminipagealign}%
1035 }%
1036 \hspace{\pcafterskip}%
1037 % tikz usage
1038 \@pc@releaseremember%
1039 \addtocounter{\@pc@global@pc@nestcnt}{-1}%
1040 \endgroup%
1041 % close spacing and potentially a single group generated by the space tester
1042 \ifthenelse{\equal{\@withinspaces}{true}}{\end@pc@withspaces}{}%
1043 \endgroup%
1044 %insert space from stacking
1045 \@pc@stackspace@forpseudocode%
1046 }
1047
1048 \newcommand{\@pc@pseudocodeminipage}[1]{%

```

```

1049 \begin{minipage}[#1]{\@pc@minipage@length}%
1050 \ifthenelse{\value{@pcsubprogstep}=0}{%
1051 \pc@display@pseudocode{\@pseudocodehead}{\@pc@thecontent}%
1052 }{% if sub procedure
1053 \pc@display@subcode{\@pseudocodehead}{\@pc@thecontent}%
1054 }%
1055 \end{minipage}%
1056 }
1057
1058
1059 \newcommand{\@pc@display@gameheader}[1]{%
1060 \tikz{\gdef\i{\thepcgamecounter}%
1061 \node[anchor=base,text depth=0pt, inner sep=0.05em,outer sep=0pt] (gamenode\i) {#1};
1062 \ifthenelse{\equal{\@withinbxgame}{true}}
1063 {\node[draw,anchor=base, above=2ex of gamenode\i] (bgamenode\i) {\@bxgameheader};}
1064 {}%
1065 }%
1066 }
1067
1068 \let\pclb\relax
1069 %
1070 \newcommand{\pc@display@pseudocode}[2]{%
1071 \ifthenelse{\equal{#1}{}}{\vspace{-\baselineskip}\@pseudocodecodesize{}}{%
1072 \ifthenelse{\equal{\@withinbxgame}{true}}
1073 {\ifthenelse{\equal{\@pc@secondheader}{true}}
1074 {\@pc@display@gameheader{#1}\addtocounter{pcgamecounter}{1}\fboxsep=1pt\fbox{\vphantom{#1}\@pc@display@
1075 {\@pc@display@gameheader{#1}}}}
1076 {#1}%
1077 \@pc@headheightskip\vspace{\pheadlinesep}\vspace{\@pseudocodeheadlinesep}\@pseudocodeheadlinecmd%
1078 \vspace{-\baselineskip}\vspace{\pcbodylinesep}\vspace{\@pseudocodebodylinesep}\@pseudocodecodesize}%
1079 \begin{flalign*}#2\end{flalign*}%
1080 }
1081
1082
1083 \newcommand{\pc@display@subcode}[2]{%
1084 \begingroup%
1085 \ifthenelse{\equal{#1}{}}{\@pc@headheightskip%
1086 \vspace{\pheadlinesep}\vspace{\@pseudocodeheadlinesep}\@pseudocodeheadlinecmd}%
1087 \vspace{\pcbodylinesep}\vspace{\@pseudocodebodylinesep}}%
1088 \@pseudocodesubcodesize%
1089 $\begin{aligned}#2\end{aligned}$%
1090 \endgroup%
1091 }
1092
1093
1094 \newcommand{\@pc@gettikzwidth}[2]{ % #1 = width, #2 = height
1095 \pgfextractx{\@tempdima}{\pgfpointdiff{\pgfpointanchor{current bounding box}{south west}}
1096 {\pgfpointanchor{current bounding box}{north east}}}
1097 \global#1=\@tempdima
1098 \pgfextracty{\@tempdima}{\pgfpointdiff{\pgfpointanchor{current bounding box}{south west}}
1099 {\pgfpointanchor{current bounding box}{north east}}}
1100 \global#2=\@tempdima
1101 }
1102
1103

```

9.7 Create Pseudocode/Procedure Commands

```

1104 %
1105 % parameter reordering
1106 \def\@pseudocodeB#1#2[#3]#4{\setkeys*{pcspace}{#2,#3}\@pseudocode[head={#1#4},#2,#3]}
1107 \def\@pseudocodeC#1#2#3{\setkeys*{pcspace}{#2}\@pseudocode[head={#1#3},#2]}
1108 %for no headers
1109 \def\@pseudocodeE#1#2[#3]{\setkeys*{pcspace}{#2,#3}\@pseudocode[head={#1},#2,#3]}
1110 \def\@pseudocodeF#1#2{\setkeys*{pcspace}{#2}\@pseudocode[head={#1},#2]}
1111 %

```

`\createprocedurecommand` Define pseudocode command with parameters:

1. name
2. code to execute after `\begingroup`
3. head prefix
4. other config

```

1112 \newcommand*{\@pc@createproc@headmode}{text}
1113 \newcommand{\createprocedurecommand}[4]{
1114 \expandafter\gdef\csname #1\endcsname{%
1115 \begingroup%
1116 \renewcommand{\@withinspaces}{false}%
1117 #2%
1118 \@ifnextchar [%]
1119   {\@pseudocodeB{#3}{#4}}
1120   {\@pseudocodeC{#3}{#4}}%
1121 }%
1122 }

```

`\createpseudocodecommand`

```

1123 \newcommand{\createpseudocodecommand}[4]{
1124 \expandafter\gdef\csname #1\endcsname{%
1125 \begingroup%
1126 \renewcommand{\@withinspaces}{false}%
1127 #2%
1128 \@ifnextchar [%]
1129   {\@pseudocodeE{#3}{#4}}
1130   {\@pseudocodeF{#3}{#4}}%
1131 }%
1132 }

```

`\createpseudocodeblock` Creates a command that has pseudocode wrapped in an `\pchstack`.

name

options for `\pchstack`

code to execute after `\begingroup`

head prefix

other config

```

1133 \newcommand{\createpseudocodeblock}[5]{
1134 \createpseudocodecommand{#1@pc}{#3}{#4}{#5}
1135 \expandafter\gdef\csname #1\endcsname{%
1136 \@ifnextchar [%]
1137   {\csname #1@@\endcsname}

```

```

1138 {\csname #1@\endcsname}
1139 }%
1140 \expandafter\gdef\csname #1@\endcsname##1{%
1141 \begin{pchstack}[#2]
1142 \csname #1@pc\endcsname{##1}
1143 \end{pchstack}
1144 }
1145 \expandafter\gdef\csname #1@@\endcsname[##1]##2{%
1146 \begin{pchstack}[#2]
1147 \csname #1@pc\endcsname[##1]{##2}
1148 \end{pchstack}
1149 }
1150 }

```

`\createprocedureblock` Creates a command that has procedure wrapped in an `\pchstack`.

name

options for `\pchstack`

code to execute after `begin`group

head prefix

other config

```

1151 \newcommand{\createprocedureblock}[5]{
1152 \createprocedurecommand{#1@pc}{#3}{#4}{#5}
1153 \expandafter\gdef\csname #1\endcsname{%
1154 \@ifnextchar[%]
1155 {\csname #1@@\endcsname}
1156 {\csname #1@\endcsname}
1157 }%
1158 \expandafter\gdef\csname #1@\endcsname##1##2{%
1159 \begin{pchstack}[#2]
1160 \csname #1@pc\endcsname{##1}{##2}
1161 \end{pchstack}
1162 }
1163 \expandafter\gdef\csname #1@@\endcsname[##1]##2##3{%
1164 \begin{pchstack}[#2]
1165 \csname #1@pc\endcsname[##1]{##2}{##3}
1166 \end{pchstack}
1167 }
1168 }

```

`\procedure` Create `\procedure` command.

`\pseudocodeblock` 1169 `\createprocedurecommand{procedure}{-}{-}`

`\procedureblock` 1170 `\createpseudocodeblock{pseudocodeblock}{center}{-}{-}`

1171 `\createprocedureblock{procedureblock}{center}{-}{-}`

9.8 Subprocedures

```

1172
1173 %
1174 % subprocedures
1175 \newcounter{@pcsubprogcnt1}
1176 \newcounter{@pcsubprogcnt1}
1177 \newcounter{@pcsubprogcnt2}
1178 \newcounter{@pcsubprogcnt2}
1179 \newcounter{@pcsubprogcnt3}

```



```

1180 \newcounter{@pcsubprogcnt3}
1181 \newcounter{@pcsubprogcnt4}
1182 \newcounter{@pcsubprogcnt4}
1183 \newcounter{@pcsubprogcnt5}
1184 \newcounter{@pcsubprogcnt5}
1185 \newcounter{@pcsubprogcnt6}
1186 \newcounter{@pcsubprogcnt6}
1187 \newcounter{@pcsubprogcnt7}
1188 \newcounter{@pcsubprogcnt7}
1189 \newcounter{@pcsubprogcnt8}
1190 \newcounter{@pcsubprogcnt8}
1191 \newcounter{@pcsubprogcnt9}
1192 \newcounter{@pcsubprogcnt9}
1193 \newcounter{@pcsubprogstep}
1194
1195 \newenvironment{subprocedure}{%
1196 \addtocounter{@pcsubprogstep}{1}%
1197 % store old counter values
1198 \setcounter{@pcsubprogcnt\the@pcsubprogstep}{\value{pclinenumber}}%
1199 \setcounter{@pcsubprogcnt\the@pcsubprogstep}{\value{pcrlinenumber}}%
1200 }{%
1201 \setcounter{pclinenumber}{\value{@pcsubprogcnt\the@pcsubprogstep}}%
1202 \setcounter{pcrlinenumber}{\value{@pcsubprogcnt\the@pcsubprogstep}}%
1203 \addtocounter{@pcsubprogstep}{-1}}
1204
1205

```

9.9 Protocols

```

1206
1207 %
1208 % send message
1209 \newcommand{\pcshortmessageoffset}{0.5cm}
1210 \newcommand{\pcdefaultmessagelength}{3.5cm}
1211 \newcommand{\pcdefaultlongmessagelength}{6cm}
1212 \newcommand{\pcbeforemessageskip}{0pt}
1213 \newcommand{\pcaftermessageskip}{10pt}
1214 \newlength{\pcmessagearrow}
1215
1216 \newcommand*{@pcsendmessagelength}{\pcdefaultmessagelength}
1217 \newcommand*{@pcsendmessagecol}{}
1218 \newcommand*{@pcsendmessagewidth}{}
1219 \newcommand*{@pcsendmessagestyle}{}
1220 \newcommand*{@pcsendmessagetop}{}
1221 \newcommand*{@pcsendmessagebottom}{}
1222 \newcommand*{@pcsendmessageright}{}
1223 \newcommand*{@pcsendmessageleft}{}
1224 \newcommand*{@pcsendmessagetopname}{t}
1225 \newcommand*{@pcsendmessagebottomname}{b}
1226 \newcommand*{@pcsendmessagerightname}{r}
1227 \newcommand*{@pcsendmessageleftname}{l}
1228 \newcommand*{@pcsendmessagetopstyle}{}
1229 \newcommand*{@pcsendmessagebottomstyle}{}
1230 \newcommand*{@pcsendmessagerightstyle}{}
1231 \newcommand*{@pcsendmessageleftstyle}{}
1232 \newcommand*{@pcsendmessagebeforeskip}{\pcbeforemessageskip}
1233 \newcommand*{@pcsendmessageafterskip}{\pcaftermessageskip}

```

```

1234
1235 \define@key{pcsendmessage}{centercol}[]{\renewcommand*\@pcsendmessagecol{#1}}
1236 \define@key{pcsendmessage}{width}[]{\renewcommand*\@pcsendmessagewidth{#1}}
1237 \define@key{pcsendmessage}{style}[]{\renewcommand*\@pcsendmessagestyle{#1}}
1238 \define@key{pcsendmessage}{length}[]{\renewcommand*\@pcsendmessagelength{#1}}
1239 \define@key{pcsendmessage}{top}[]{\renewcommand*\@pcsendmessagetop{#1}}
1240 \define@key{pcsendmessage}{bottom}[]{\renewcommand*\@pcsendmessagebottom{#1}}
1241 \define@key{pcsendmessage}{right}[]{\renewcommand*\@pcsendmessageright{#1}}
1242 \define@key{pcsendmessage}{left}[]{\renewcommand*\@pcsendmessageleft{#1}}
1243 \define@key{pcsendmessage}{topname}[]{\renewcommand*\@pcsendmessagetopname{#1}}
1244 \define@key{pcsendmessage}{bottomname}[]{\renewcommand*\@pcsendmessagebottomname{#1}}
1245 \define@key{pcsendmessage}{rightname}[]{\renewcommand*\@pcsendmessagerightname{#1}}
1246 \define@key{pcsendmessage}{leftname}[]{\renewcommand*\@pcsendmessageleftname{#1}}
1247 \define@key{pcsendmessage}{topstyle}[]{\renewcommand*\@pcsendmessagetopstyle{#1}}
1248 \define@key{pcsendmessage}{bottomstyle}[]{\renewcommand*\@pcsendmessagebottomstyle{#1}}
1249 \define@key{pcsendmessage}{rightstyle}[]{\renewcommand*\@pcsendmessagerightstyle{#1}}
1250 \define@key{pcsendmessage}{leftstyle}[]{\renewcommand*\@pcsendmessageleftstyle{#1}}
1251 \define@key{pcsendmessage}{beforeskip}[]{\renewcommand*\@pcsendmessagebeforeskip{#1}}
1252 \define@key{pcsendmessage}{afterskip}[]{\renewcommand*\@pcsendmessageafterskip{#1}}
1253
1254 \newcommand*\@pcsendmessagealignedtop{false}
1255 \define@key{pcsendmessage}{topaligned}[true]{\renewcommand*\@pcsendmessagealignedtop{#1}}
1256 \newcommand*\@pcsendmessagealignedbottom{false}
1257 \define@key{pcsendmessage}{bottomaligned}[true]{\renewcommand*\@pcsendmessagealignedbottom{#1}}
1258 \newcommand*\@pcsendmessagealignedleft{false}
1259 \define@key{pcsendmessage}{leftaligned}[true]{\renewcommand*\@pcsendmessagealignedleft{#1}}
1260 \newcommand*\@pcsendmessagealignedright{false}
1261 \define@key{pcsendmessage}{rightaligned}[true]{\renewcommand*\@pcsendmessagealignedright{#1}}
1262
1263
1264 \newcommand{\@pc@centerincol}[2]{%
1265 \ifmeasuring%
1266 #2%
1267 \else%
1268 \makebox[\ifcase\expandafter #1\maxcolumn@widths\fi]{\displaystyle#2$}%
1269 \fi%
1270 }
1271
1272 \newcommand{\@pc@centerincol}[1]{\@pc@centerincol{\thepccolumncounter}{#1}}
1273
1274 \newcommand{\@do@sendmessage}[1]{%
1275 \ifthenelse{\equal{\@pcsendmessagecol}{}}{%
1276 \ifthenelse{\equal{\@pcsendmessagewidth}{}}{#1}{% we have some width
1277 \makebox[\@pcsendmessagewidth]{\displaystyle#1$}%
1278 }}{%we know the column to center on
1279 \@pc@centerincol{\@pcsendmessagecol}{#1}%
1280 }%
1281 }
1282
1283 \newcommand*\@sendmessage}[2]{%
1284 \begingroup\setkeys{pcsendmessage}{#2}%
1285 \tikzset{PCSENDMSG-PATH-STYLE/.style/.expand once=\@pcsendmessagestyle}%
1286 \tikzset{PCSENDMSG-TOP-STYLE/.style/.expand once=\@pcsendmessagetopstyle}%
1287 \tikzset{PCSENDMSG-BOTTOM-STYLE/.style/.expand once=\@pcsendmessagebottomstyle}%
1288 \tikzset{PCSENDMSG-LEFT-STYLE/.style/.expand once=\@pcsendmessageleftstyle}%
1289 \tikzset{PCSENDMSG-RIGHT-STYLE/.style/.expand once=\@pcsendmessagerightstyle}%

```

```

1290 %
1291 %
1292 \ifthenelse{\equal{\@pcsendmessagealignedtop}{true}}
1293 {\ifthenelse{\equal{\@pcsendmessagetop}{}}
1294 {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%
1295 {\newcommand{\@pc@fin@sendmessagetop}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagetop\end{aligned}}}
1296 {\let\@pc@fin@sendmessagetop\@pcsendmessagetop}%
1297 %
1298 \ifthenelse{\equal{\@pcsendmessagealignedbottom}{true}}
1299 {\ifthenelse{\equal{\@pcsendmessagebottom}{}}
1300 {\let\@pc@fin@sendmessagebottom\@pcsendmessagebottom}%
1301 {\newcommand{\@pc@fin@sendmessagebottom}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessagebottom\end{aligned}}}
1302 {\let\@pc@fin@sendmessagebottom\@pcsendmessagebottom}%
1303 %
1304 \ifthenelse{\equal{\@pcsendmessagealignedright}{true}}
1305 {\ifthenelse{\equal{\@pcsendmessageright}{}}
1306 {\let\@pc@fin@sendmessageright\@pcsendmessageright}
1307 {\newcommand{\@pc@fin@sendmessageright}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessageright\end{aligned}}}
1308 {\let\@pc@fin@sendmessageright\@pcsendmessageright}%
1309 %
1310 \ifthenelse{\equal{\@pcsendmessagealignedleft}{true}}
1311 {\ifthenelse{\equal{\@pcsendmessageleft}{}}
1312 {\let\@pc@fin@sendmessageleft\@pcsendmessageleft}
1313 {\newcommand{\@pc@fin@sendmessageleft}{\let\halign\@pc@halign$\begin{aligned}\@pcsendmessageleft\end{aligned}}}
1314 {\let\@pc@fin@sendmessageleft\@pcsendmessageleft}%
1315 %restore halign
1316 %
1317 \addtocounter{\@pcsubprogstep}{1}%
1318 \hspace{\@pcsendmessagebeforeskip}%
1319 \begin{varwidth}{\linewidth}
1320 \do@sendmessage{
1321 \begin{tikzpicture}%
1322 \node[PCSENDMSG-LEFT-STYLE] (\@pcsendmessageleftname) {\@pc@fin@sendmessageleft};
1323 \node[right=\@pcsendmessagelength of \@pcsendmessageleftname,PCSENDMSG-RIGHT-STYLE] (\@pcsendmessagerightname) {\@pc@fin@sendmessageright};
1324 \path[#1,PCSENDMSG-PATH-STYLE] (\@pcsendmessageleftname) edge[] node[above,PCSENDMSG-TOP-STYLE] (\@pcsendmessagetopname) (\@pcsendmessagerightname);
1325 \end{tikzpicture}%
1326 }%
1327 \end{varwidth}
1328 \addtocounter{\@pcsubprogstep}{-1}%
1329 \hspace{\@pcsendmessageafterskip}%
1330 \endgroup%
1331 }
1332
1333 \WithSuffix\newcommand\sendmessage*[2]{%
1334 \sendmessage{#1}{topaligned,leftaligned,bottomaligned,rightaligned,#2}%
1335 }
1336
1337 \newcommandx*\sendmessageright*[2][1=->]{%
1338 \sendmessage{#1}{#2}%
1339 }
1340
1341 \newcommandx*\sendmessageleft*[2][1=<-]{%
1342 \sendmessage{#1}{#2}%
1343 }
1344
1345 \WithSuffix\newcommand\sendmessageleft*[2][\pcdefaultmessagelength]{%

```

```

1346 \begingroup%
1347 \renewcommand{\@pcsendmessagetop}{\let\halign\@pc@halign$\begin{aligned}#2\end{aligned}$}%
1348 \sendmessage{<-}{length=#1}%
1349 \endgroup%
1350 }
1351
1352
1353 \WithSuffix\newcommand\sendmessageright*[2][\pcdefaultmessagelength]{%
1354 \begingroup%
1355 \renewcommand{\@pcsendmessagetop}{\let\halign\@pc@halign$\begin{aligned}#2\end{aligned}$}%
1356 \sendmessage{->}{length=#1}%
1357 \endgroup%
1358 }
1359
1360 \WithSuffix\newcommand\sendmessagerightleft*[2][\pcdefaultmessagelength]{%
1361 \begingroup%
1362 \renewcommand{\@pcsendmessagetop}{\let\halign\@pc@halign$\begin{aligned}#2\end{aligned}$}%
1363 \sendmessage{<->}{length=#1}%
1364 \endgroup%
1365 }
1366
1367 \DeclareExpandableDocumentCommand\sendmessagerightx{0{\pcdefaultlongmessagelength}m0{m}}{%
1368 \multicolumn{#2}{c}{\ensuremath{\hspace{\pcbeforemessageskip}\xrightarrow[\begin{aligned}#3\end{aligned}]{}}}
1369 }
1370
1371 \DeclareExpandableDocumentCommand\sendmessageleftx{0{\pcdefaultlongmessagelength}m0{m}}{%
1372 \multicolumn{#2}{c}{\ensuremath{\hspace{\pcbeforemessageskip}\xleftarrow[\begin{aligned}#3\end{aligned}]{}}}
1373 }
1374
1375 %
1376 % Division
1377 \DeclareExpandableDocumentCommand{\pcintertext}{0{m}}{\intertext{%
1378 \ifthenelse{equal{#1}{center}}{\makebox[\linewidth][c]{#2}}{%
1379 \ifthenelse{equal{#1}{dotted}}{\dotfill#2\dotfill}{}%
1380 \ifthenelse{equal{#1}{}}{#2}{}%
1381 }\@pc@beginnewline}
1382
1383
1384

```

9.10 Tikz within Pseudocode

```

1385
1386 %
1387 % remember pictues
1388 \newcounter{@pc@remember}
1389
1390 \newcommand{\@pc@ensureremember}{%
1391 \ifthenelse{value{@pc@remember}=0}{\tikzstyle{every picture}+=[remember picture]}{%
1392 \addtocounter{@pc@remember}{1}}
1393
1394 \newcommand{\@pc@releaseremember}{%
1395 \addtocounter{@pc@remember}{-1}%
1396 \ifthenelse{value{@pc@remember}=0}{\tikzstyle{every picture}-=[remember picture]}{%
1397 }
1398
1399

```

```

1400 %
1401 % pcimage
1402 \newenvironment{pcimage}{%
1403 \begin{group}\@pc@ensureremember%
1404 }{%
1405 \@pc@releaseremember\endgroup%
1406 }
1407
1408 \newcommand*\@pcnodecontent{}
1409 \newcommand*\@pcnodestyle{}
1410 \newcommand*\@pcnodedraw{}
1411 \define@key{pcnode}{content}[]{\renewcommand*\@pcnodecontent{#1}}
1412 \define@key{pcnode}{style}[]{\renewcommand*\@pcnodestyle{#1}}
1413 \define@key{pcnode}{draw}[]{\renewcommand*\@pcnodedraw{#1}}
1414
1415 \newcommandx*\pcnode[2][2=]{%
1416 \begin{group}\setkeys{pcnode}{#2}%
1417 \tikzset{PCNODE-STYLE/.style/.expand once=\@pcnodestyle}%
1418 \begin{tikzpicture}[inner sep=0ex,baseline=0pt]%
1419 \node[PCNODE-STYLE] (#1) {\@pcnodecontent}; %
1420 \end{tikzpicture}%
1421 \ifdefempty{\@pcnodedraw}{\@pcnodedraw}{%
1422 \begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt]\@pcnodedraw\end{tikzpicture}
1423 }%
1424 \endgroup}
1425
1426 \newcommandx*\pcdraw[2][2=]{%
1427 \begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt,#2]
1428 #1
1429 \end{tikzpicture}}
1430

```

9.11 Black Box Reductions

```

1431
1432 %
1433 % Reductions
1434 \newcommand{\@bb@lastbox}{}
1435 \newcommand{\@bb@lastoracle}{}
1436 \newcommand{\@bb@lastchallenger}{}
1437
1438 \newlength{\@bb@message@voffset}
1439 \newlength{\@bb@query@voffset}
1440 \newlength{\@bb@oraclequery@voffset}
1441 \newlength{\@bb@challengerquery@voffset}
1442
1443 \newcounter{\@bb@oracle@cnt}
1444 \newcounter{\@bb@oracle@nestcnt}
1445 \newcounter{\@bb@challenger@cnt}
1446 \newcounter{\@bb@challenger@nestcnt}
1447
1448 \newcounter{\@bb@env@nestcnt}
1449
1450 \newcommand{\bb@oraclenodenameprefix}{ora-}
1451 \newcommand{\bb@rchallengernodenameprefix}{challenger-}
1452 \newcommand{\bb@envnodenameprefix}{env-}

```

aboveskip
\@pc@bbrenvaboveskip 1453 \newcommand*\@pc@bbrenvaboveskip{Opt}
1454 \define@key{pcbbrenv}{aboveskip}[Opt]{\renewcommand*\@pc@bbrenvaboveskip{#1}}

belowskip
\@pc@bbrenvbelowskip 1455 \newcommand*\@pc@bbrenvbelowskip{Opt}
1456 \define@key{pcbbrenv}{belowskip}[Opt]{\renewcommand*\@pc@bbrenvbelowskip{#1}}

bbrenv@legacyargcheck ensures that first command can still be 5cm which is rewritten as aboveskip=5cm
\@pc@bbrenv@argstring 1457 \newcommand*\@pc@bbrenv@argstring{
1458 \def\@pc@bbrenv@remfinalequals#1=#2=\relax{\renewcommand*\@pc@bbrenv@argstring{#1=#2}}
1459 \def\@pc@bbrenv@legacyargcheck#1=#2\relax{%
1460 \ifthenelse{\equal{#2}{}}
1461 {\PackageWarning{cryptocode}{Deprecated option for bbrenv. Please use key value list as first parameter}}
1462 \renewcommand*\@pc@bbrenv@argstring{aboveskip=#1}}
1463 {\@pc@bbrenv@remfinalequals#1=#2\relax}%
1464 }

\bbrfirstmessageoffset offset of the first message from top
1465 \providecommand{\bbrfirstmessageoffset}{1ex}

tikzargs Allow passing in arguments to tikzpicture.
\bbrtikzargs 1466 \newcommand*\bbrtikzargs{
1467 \define@key{pcbbrenv}{tikzargs}[]{\renewcommand*\bbrtikzargs{#1}}

bbrenv Black Box Reduction Environment
1468 \newenvironmentx{bbrenv}[3][1={aboveskip=0pt,belowskip=0pt},3=0pt]{%
1469 \addtocounter{@bb@env@nestcnt}{1}%
1470 \renewcommand{\@bb@lastbox}{#2}%
1471 % parse args and allow old style #1=0pt
1472 \@pc@bbrenv@legacyargcheck#1=\relax%
1473 \@expandedsetkeys{pcbbrenv}{\belowskip=#3}{\@pc@bbrenv@argstring}{}%
1474 %
1475 % reset lengths
1476 \@pc@globalsetlength{\@bb@message@voffset}{\bbrfirstmessageoffset}%
1477 \@pc@globalsetlength{\@bb@query@voffset}{\bbrfirstmessageoffset}%
1478 \@pc@globalsetlength{\@bb@oraclequery@voffset}{\bbrfirstmessageoffset}%
1479 \@pc@globalsetlength{\@bb@challengerquery@voffset}{\bbrfirstmessageoffset}%
1480 %
1481 %reset oracle counter and oracle query offset
1482 \ifthenelse{\value{@bb@oracle@nestcnt}=0}
1483 {\setcounter{@bb@oracle@cnt}{0}}{ }%
1484 \ifthenelse{\value{@bb@challenger@nestcnt}=0}
1485 {\setcounter{@bb@challenger@cnt}{0}}{ }%
1486 %
1487 \vspace{\@pc@bbrenvaboveskip}%
1488 \ifthenelse{\value{@bb@env@nestcnt}=1}
1489 {\@pc@ensureremember%
1490 \begin{tikzpicture}[baseline=0pt,\bbrtikzargs]
1491 }\tikz\bggroup
1492 }{ }%
1493 \ifthenelse{\value{@bb@env@nestcnt}=1}
1494 {\end{tikzpicture}}%
1495 \@pc@releaseremember%
1496 }\egroup}%

```

1497 \vspace{\@pc@bbrenvbelowskip}%
1498 \addtocounter{@bb@env@nestcnt}{-1}%
1499 % reset lengths
1500 \@pc@globalsetlength{\@bb@message@voffset}{\bbrfirstmessageoffset}%
1501 \@pc@globalsetlength{\@bb@query@voffset}{\bbrfirstmessageoffset}%
1502 \@pc@globalsetlength{\@bb@oraclequery@voffset}{\bbrfirstmessageoffset}%
1503 \@pc@globalsetlength{\@bb@challengerquery@voffset}{\bbrfirstmessageoffset}%
1504 }

```

black box reduction box option keys

```

1505 \newcommand*\bbrboxname{}
1506 \newcommand*\bbrboxnamepos{right}
1507 \newcommand*\bbrboxnamestyle{}
1508 \newcommand*\@bbrboxnamepos{below right=0.5ex and -0.5ex of \@bb@lastbox.north east,anchor=north east}
1509 \newcommand*\bbrboxabovesep{\baselineskip}
1510 \newcommand*\@bbrboxnameposoffset{below left=\bbrboxabovesep of phantomname.south west}
1511 \newcommand*\bbrboxstyle{draw}
1512 \newcommand*\bbrboxafterskip{}
1513 \newcommand*\bbrboxminheight{0pt}
1514 \newcommand*\bbrboxminwidth{2cm}
1515 \newcommand*\bbrboxxshift{0pt}
1516 \newcommand*\bbrboxyshift{0pt}
1517 \define@key{bbrbox}{abovesep}[]{\renewcommand*\bbrboxabovesep{#1}}
1518 \define@key{bbrbox}{name}[]{\renewcommand*\bbrboxname{#1}}
1519 \define@key{bbrbox}{namestyle}[]{\renewcommand*\bbrboxnamestyle{#1}}
1520 \define@key{bbrbox}{namepos}[]{\renewcommand*\bbrboxnamepos{#1}}
1521 \define@key{bbrbox}{style}[draw]{\renewcommand*\bbrboxstyle{#1}}
1522 \define@key{bbrbox}{minwidth}[]{\renewcommand*\bbrboxminwidth{#1}}
1523 \define@key{bbrbox}{addheight}[]{\renewcommand*\bbrboxafterskip{#1}}
1524 \define@key{bbrbox}{minheight}[]{\renewcommand*\bbrboxminheight{#1}}
1525 \define@key{bbrbox}{xshift}[]{\renewcommand*\bbrboxxshift{#1}}
1526 \define@key{bbrbox}{yshift}[]{\renewcommand*\bbrboxyshift{#1}}
1527
1528
1529 \NewEnviron{bbrbox}[1][]{%
1530 \setkeys{bbrbox}{#1}%
1531
1532 \ifthenelse{\equal{\bbrboxnamepos}{center}}
1533 {\renewcommand{\@bbrboxnamepos}{below=0.5ex of \@bb@lastbox.north,anchor=north}}{}
1534 \ifthenelse{\equal{\bbrboxnamepos}{left}}
1535 {\renewcommand{\@bbrboxnamepos}{below=0.5ex of \@bb@lastbox.north west,anchor=north west}}{}
1536 \ifthenelse{\equal{\bbrboxnamepos}{top right}}
1537 {\renewcommand{\@bbrboxnamepos}{above=0cm of \@bb@lastbox.north east,anchor=south east}\renewcommand{
1538 \ifthenelse{\equal{\bbrboxnamepos}{top center}}
1539 {\renewcommand{\@bbrboxnamepos}{above=0cm of \@bb@lastbox.north,anchor=south}\renewcommand{\@bbrboxna
1540 \ifthenelse{\equal{\bbrboxnamepos}{top left}}
1541 {\renewcommand{\@bbrboxnamepos}{above=0cm of \@bb@lastbox.north west,anchor=south west}\renewcommand{
1542 \ifthenelse{\equal{\bbrboxnamepos}{middle}}
1543 {\renewcommand{\@bbrboxnamepos}{above=0.5ex of \@bb@lastbox.base,anchor=south}}{}
1544 \ifthenelse{\equal{\bbrboxnamepos}{bottom}}
1545 {\renewcommand{\@bbrboxnamepos}{above=0.5ex of \@bb@lastbox.base,anchor=north}}{}
1546
1547
1548 \tikzset{BBRBOXSTYLE/.style/.expand once=\bbrboxstyle}%
1549 \tikzset{BBRBOXNAMEPOS/.style/.expand once=\bbrboxnamepos}%
1550 \tikzset{BBRBOXNAMESTYLE/.style/.expand once=\bbrboxnamestyle}%

```

```

1551 \tikzset{BBRBOXNAMEPOSOFFSET/.style/.expand once=\@bbrboxnameposoffset}%
1552
1553 \ifthenelse{\equal{\bbrboxxshift}{}} \OR \equal{\bbrboxxshift}{0pt}}{
1554 \coordinate[inner sep=0pt,outer sep=0pt] (\@bb@lastbox-tmpouter) {};
1555 }{
1556 \node[inner sep=0pt, outer sep=0pt] (\@bb@lastbox-tmpouter) {}; %this empty node seems needed to get tl
1557 }
1558
1559 \node[inner sep=.3333em,anchor=north,BBRBOXSTYLE,minimum height=\bbrboxminheight,below right=\bbrboxysl
1560 \tikz{
1561 \node[inner sep=0pt,outer sep=0pt,minimum height=0cm] (phantomname) {}; %minimum width
1562 \node[BBRBOXNAMEPOSOFFSET,minimum height=0cm] (\@bb@lastbox-inner) {\begin{varwidth}{2\linewidth}\BODY
1563 \ifthenelse{\equal{\bbrboxafterskip}{}}{}{}{
1564 \node[below=0cm of \@bb@lastbox-inner,minimum height=\bbrboxafterskip] {};
1565 }
1566 \node[inner sep=0pt,outer sep=0pt,at=(\@bb@lastbox-inner.south west),minimum height=0cm] () {\phantom{
1567 }
1568 \egroup;
1569 \ifthenelse{\equal{\bbrboxnamepos}{none}}
1570 {}{\node[BBRBOXNAMEPOS,BBRBOXNAMESTYLE, inner sep=0.2ex, outer sep=0pt, overlay] () {\bbrboxname};}
1571 }
1572
1573
1574 \newcommand*\bbroraclevdistance{\baselineskip}
1575 \newcommand*\bbroraclehdistance{1.5cm}
1576 \define@key{bbroracle}{distance}[]{\renewcommand*\bbroraclehdistance{#1}}
1577 \define@key{bbroracle}{hdistance}[]{\renewcommand*\bbroraclehdistance{#1}}
1578 \define@key{bbroracle}{vdistance}[]{\renewcommand*\bbroraclevdistance{#1}}
1579
1580
1581 % ORACLES
1582 \newenvironment{x{bbroracle}[2][2=]}{
1583 \begin{group}
1584 \setkeys{bbroracle}{#2}
1585 %reset query boolean. This is a bit crude and does not allow nesting oracles
1586 %in oracles but should be good enough
1587 \gdef\@bbr@first@oraclequery{true}
1588 %add to nesting cout
1589 \addtocounter{\@bb@oracle@nestcnt}{1}
1590 %if first oracle, then put it to the right, else stack them vertically
1591 \addtocounter{\@bb@oracle@cnt}{1}
1592 \ifthenelse{\value{\@bb@oracle@cnt}=1}{
1593 \setlength{\@bb@tmplength@b}{\bbroraclevdistance-\baselineskip}
1594 \node[inner sep=0pt,below right=\@bb@tmplength@b and \bbroraclehdistance of \@bb@lastbox.north east,an
1595 }{
1596 % compute distance of top of last box to bottom of last oracle
1597 \coordinate (@bbtmpcoord) at (\@bb@lastbox.north east);
1598 \path (@bbtmpcoord);
1599 \pgfgetlastxy{\XCoord}{\YCoordA}
1600 \coordinate (@bbtmpcoord) at (\bbroraclenodenameprefix \@bb@lastoracle.south west);
1601 \path (@bbtmpcoord);
1602 \pgfgetlastxy{\XCoord}{\YCoordB}
1603 \setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB+\bbroraclevdistance}
1604 \node[inner sep=0pt,below right=\@bb@tmplength@b and \bbroraclehdistance of \@bb@lastbox.north east,an
1605 }
1606 \global\def\@bb@lastoracle{#1}

```



```

1607 \begin{bbrenv}{#1}
1608 }{
1609 \end{bbrenv}
1610 \egroup;
1611
1612 \addtocounter{@bb@oracle@nestcnt}{-1}
1613 \endgroup
1614 }
1615
1616
1617 \newcommand*\bbrchallengerhdistance{1.5cm}
1618 \newcommand*\bbrchallengervdistance{\baselineskip}
1619 \define@key{bbrchallenger}{distance}[]{\renewcommand*\bbrchallengerhdistance{#1}}
1620 \define@key{bbrchallenger}{hdistance}[]{\renewcommand*\bbrchallengerhdistance{#1}}
1621 \define@key{bbrchallenger}{vdistance}[]{\renewcommand*\bbrchallengervdistance{#1}}
1622
1623
1624 % Challenger
1625 \newenvironmentx{bbrchallenger}[2][2=]{%
1626 \begingroup%
1627 \setkeys{bbrchallenger}{#2}%
1628 %reset query boolean. This is a bit crude and does not allow nesting oracles
1629 %in oracles but should be good enough
1630 \gdef\@bbr@first@challengerquery{true}%
1631 %add to nesting cout
1632 \addtocounter{@bb@challenger@nestcnt}{1}%
1633 %if first oracle, then put it to the right, else stack them vertically
1634 \addtocounter{@bb@challenger@cnt}{1}%
1635 \ifthenelse{\value{@bb@challenger@cnt}=1}{%
1636 \setlength{\@bb@tmplength@b}{\bbrchallengervdistance-\baselineskip}%
1637 \node[inner sep=0pt,outer sep=0pt,below left=\@bb@tmplength@b and \bbrchallengerhdistance of \@bb@last]
1638 }{%
1639 \coordinate (@bbtmpcoord) at (\@bb@lastbox.north west);%
1640 \path (@bbtmpcoord);%
1641 \pgfgetlastxy{\XCoord}{\YCoordA}%
1642 \coordinate (@bbtmpcoord) at (\bbrchallengernodenameprefix \@bb@lastchallenger.south east);%
1643 \path (@bbtmpcoord);%
1644 \pgfgetlastxy{\XCoord}{\YCoordB}%
1645 \setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB+\bbrchallengervdistance}%
1646 \node[inner sep=0pt,below left=\@bb@tmplength@b and \bbrchallengerhdistance of \@bb@lastbox.north west]
1647 }{%
1648 \global\def\@bb@lastchallenger{#1}
1649 \begin{bbrenv}{#1}%
1650 }{
1651 \end{bbrenv}%
1652 \egroup;%
1653 \addtocounter{@bb@challenger@nestcnt}{-1}%
1654 \endgroup%
1655 \let\msgfrom\bbrchallengerqueryto%
1656 }
1657
1658
1659 \newcommand*\bbrinputlength{0.5cm}
1660 \newcommand*\bbrinputhoffset{0.5cm}
1661 \newcommand*\bbrinputbottom{}
1662 \newcommand*\bbrinputtop{}

```

```

1663 \newcommand*\bbrinputedgestyle{}
1664 \newcommand*\bbrinputtopstyle{}
1665 \newcommand*\bbrinputbottomstyle{}
1666 \newcommand*\bbrinputnodestyle{}
1667 \newcommand*\bbrinputnodename{}
1668 \define@key{bbrinput}{length}[]{\renewcommand*\bbrinputlength{#1}}
1669 \define@key{bbrinput}{hoffset}[]{\renewcommand*\bbrinputhoffset{#1}}
1670 \define@key{bbrinput}{name}[]{\renewcommand*\bbrinputnodename{#1}}
1671 \define@key{bbrinput}{top}[]{\renewcommand*\bbrinputtop{#1}}
1672 \define@key{bbrinput}{bottom}[]{\renewcommand*\bbrinputbottom{#1}}
1673
1674
1675 \newcommand{\@bb@inputsetup}[1]{
1676 %load keys
1677 \begingroup % for local keys
1678
1679 \setkeys{bbrinput}{#1}%
1680
1681 \tikzset{BBRINPUT-NODESTYLE/.style/.expand once=\bbrinputedgestyle}%
1682 \tikzset{BBRINPUT-TOPSTYLE/.style/.expand once=\bbrinputtopstyle}%
1683 \tikzset{BBRINPUT-BOTTOMSTYLE/.style/.expand once=\bbrinputbottomstyle}%
1684 \tikzset{BBRINPUT-EDGESTYLE/.style/.expand once=\bbrinputedgestyle}%
1685
1686 }
1687
1688 \newcommand{\@bb@inputfinalize}{
1689 \endgroup
1690 }
1691
1692 \newcommandx*\{bbrinput}[2][2=]{%
1693 \@bb@inputsetup{#2}
1694 \ifthenelse{\equal{\bbrinputnodename}{}}{
1695   {\renewcommand{\bbrinputnodename}{\@bb@lastbox-input}}{}
1696
1697 \node[overlay,above right={\bbrinputlength} and {\bbrinputhoffset} of \@bb@lastbox.north west, anchor=
1698 \path[->] (\bbrinputnodename.south) edge[BBRINPUT-EDGESTYLE] node[above,anchor=east,BBRINPUT-TOPSTYLE
1699 \@bb@inputfinalize
1700 }
1701
1702 \newcommandx*\{bbroutput}[2][2=]{%
1703 \@bb@inputsetup{#2}
1704 \ifthenelse{\equal{\bbrinputnodename}{}}{
1705   {\renewcommand{\bbrinputnodename}{\@bb@lastbox-output}}{}
1706
1707 \node[overlay,below right={\bbrinputlength} and {\bbrinputhoffset} of \@bb@lastbox.south west, anchor=
1708 \draw[->] (\bbrinputnodename.north|-\@bb@lastbox.south) -- (\bbrinputnodename.north|-\bbrinputnodename
1709 \@bb@inputfinalize
1710 }
1711
1712 \newenvironment{bbrpic}[1][]{%
1713 \begin{tikzpicture}[overlay,inner sep=0ex,baseline=0pt,#1]%
1714 }{%
1715 \end{tikzpicture}}
1716
1717 %
1718 % communication

```

```

1719 %temporary lengths
1720 \newlength{\@bb@com@tmpoffset}
1721 \newlength{\@bb@tmplength@b}
1722
1723 %keys
1724 \newcommand*\@bbrcomsidestyle{}
1725 \newcommand*\@bbrcomosidestyle{}
1726 \newcommand*\@bbrcomtopstyle{}
1727 \newcommand*\@bbrcombottstyle{}
1728 \newcommand*\@bbrcomside{}
1729 \newcommand*\@bbrcomoside{}
1730 \newcommand*\@bbrcomtop{}
1731 \newcommand*\@bbrcombott{}
1732 \newcommand*\@bbrcomedgestyle{}
1733 \newcommand*\@bbrcomlength{1.25cm}
1734 \newcommand*\@bbrcomtopname{bbrcomtop}
1735 \newcommand*\@bbrcombottname{bbrcombott}
1736 \newcommand*\@bbrcomsidename{bbrcomside}
1737 \newcommand*\@bbrcomosidenam{bbrcomoside}
1738 \newcommand*\@bbrcombeforeskip{0pt}
1739 \newcommand*\@bbrcomafterskip{0ex}
1740 \define@key{bbrcom}{sidestyle}[]{\renewcommand*\@bbrcomsidestyle{#1}}
1741 \define@key{bbrcom}{osidestyle}[]{\renewcommand*\@bbrcomosidestyle{#1}}
1742 \define@key{bbrcom}{topstyle}[]{\renewcommand*\@bbrcomtopstyle{#1}}
1743 \define@key{bbrcom}{bottstyle}[]{\renewcommand*\@bbrcombottstyle{#1}}
1744 \define@key{bbrcom}{side}[]{\renewcommand*\@bbrcomside{#1}}
1745 \define@key{bbrcom}{oside}[]{\renewcommand*\@bbrcomoside{#1}}
1746 \define@key{bbrcom}{top}[]{\renewcommand*\@bbrcomtop{#1}}
1747 \define@key{bbrcom}{bott}[]{\renewcommand*\@bbrcombott{#1}}
1748 \define@key{bbrcom}{edgestyle}[]{\renewcommand*\@bbrcomedgestyle{#1}}
1749 \define@key{bbrcom}{length}[]{\renewcommand*\@bbrcomlength{#1}}
1750 \define@key{bbrcom}{topname}[]{\renewcommand*\@bbrcomtopname{#1}}
1751 \define@key{bbrcom}{bottname}[]{\renewcommand*\@bbrcombottname{#1}}
1752 \define@key{bbrcom}{sidename}[]{\renewcommand*\@bbrcomsidename{#1}}
1753 \define@key{bbrcom}{osidenam}[]{\renewcommand*\@bbrcomosidenam{#1}}
1754 \define@key{bbrcom}{beforeskip}[]{\renewcommand*\@bbrcombeforeskip{#1}}
1755 \define@key{bbrcom}{aboveskip}[]{\renewcommand*\@bbrcombeforeskip{#1}}
1756 \define@key{bbrcom}{afterskip}[]{\renewcommand*\@bbrcomafterskip{#1}}
1757 \define@key{bbrcom}{belowskip}[]{\renewcommand*\@bbrcomafterskip{#1}}

```

\@bbrcomfixedoffset Provide means for fixed message offset from top or bottom

```

\@bbrcomfixedboffset
    fixedoffset
    fixedboffset
1758 \newcommand*\@bbrcomfixedoffset{}
1759 \newcommand*\@bbrcomfixedboffset{false}
1760 \define@key{bbrcom}{fixedoffset}[]{\renewcommand*\@bbrcomfixedoffset{#1}}
1761 \define@key{bbrcom}{fixedboffset}[]{\renewcommand*\@bbrcomfixedoffset{#1}\renewcommand*\@bbrcomfixedboffset{#1}}

1762 %
1763 %
1764 \newcommand*\@bbrbasenodestyle{}
1765 \newcommand*\@bbrbasenodename{bbrtmpname}
1766 \define@key{bbrabase}{nodestyle}[]{\renewcommand*\@bbrbasenodestyle{#1}}
1767 \define@key{bbrabase}{nodename}[]{\renewcommand*\@bbrbasenodename{#1}}
1768
1769 \newcommand*\@bbr@first@msg{true}
1770 \newcommand*\@bbr@first@query{true}
1771 \newcommand*\@bbr@first@oraclequery{true}

```

```

1772 \newcommand*\@bbr@first@challengerquery{true}
1773
\@bbr@intermessage@skip Skip between two messages.
\@bbr@intermessage@medskip 1774 \newcommand*\@bbr@intermessage@skip{4ex}
\@bbr@intermessage@shortskip 1775 \newcommand*\@bbr@intermessage@veryshortskip{1ex}
\@bbr@intermessage@veryshortskip 1776 \newcommand*\@bbr@intermessage@shortskip{1.5ex}
1777 \newcommand*\@bbr@intermessage@medskip{2.5ex}

islast Sets the message from the bottom of the box with the same distance as the first message.
\@bbrcomislast 1778 \newcommand*\@bbrcomislast{false}
1779 \define@key{bbrcom}{islast}[true]{\renewcommand*\@bbrcomislast{#1}}
1780
1781 \newcommand*\@bbrcom@check@islast{%
1782 \ifthenelse{\equal{\@bbrcomislast}{true}}
1783 {\renewcommand*\@bbrcomfixedoffset{\bbrfirstmessageoffset}\renewcommand*\@bbrcomfixedboffset{true}}
1784 {}
1785 }

\@bbr@lastskip marker to set whether next skip is a short or a long one
1786 \def\@bbr@lastskip{0pt}

\@bb@comsetup Sets up communication parameters for message/query commands. Parameters are {\key
value list}, {\length}, {\command for adding space} {\true if first message}
1787 \newcommand{\@bb@comsetup}[4]{
1788 % check if is first message and mark as false
1789 \edef\@tmp@bbr@isfirst{#4}
1790 \renewcommand#4{false}
1791
1792 %load keys
1793 \begingroup % for local keys
1794
1795 \setkeys{bbrcom}{#1}%
1796
1797 %set styles
1798 \tikzset{BBRCOM-SIDESTYLE/.style/.expand once=\@bbrcomsidestyle}%
1799 \tikzset{BBRCOM-OSIDESTYLE/.style/.expand once=\@bbrcomosidestyle}%
1800 \tikzset{BBRCOM-TOPSTYLE/.style/.expand once=\@bbrcomtopstyle}%
1801 \tikzset{BBRCOM-BOTTOMSTYLE/.style/.expand once=\@bbrcombottstyle}%
1802 \tikzset{BBRCOM-EDGESTYLE/.style/.expand once=\@bbrcomedgestyle}%
1803
1804 \@bbrcom@check@islast{}
1805
1806 % increase space
1807 #3{\@bbrcombeforeskip}
1808 \ifthenelse{\equal{\@bbrcomfixedoffset}{}}
1809 {
1810 \ifthenelse{\equal{\@tmp@bbr@isfirst}{true}}
1811 {}{#3{\@bbr@lastskip}}
1812
1813 \setlength{\@bb@com@tmpoffset}{#2}%
1814 }
1815 {
1816 \setlength{\@bb@com@tmpoffset}{\@bbrcomfixedoffset}%
1817 }
1818 }

```

\@bb@comfinalize

```
1819 \newcommand{\@bb@comfinalize}[1]{
1820 #1{\@bbrcomafterskip}
1821 \endgroup
1822 \def\@bbr@lastskip{\@bbr@intermessage@skip}
1823 }
```

\bbrmsg 9 -> true if first message 10 -> anchor from bottom

```
1824 \newcommand{\bbrmsg}[9]{
1825 \@bb@comsetup{#1}{#7}{#8}{#9}
1826 %
1827 \ifthenelse{\equal{\@bbrcomfixedboffset}{true}}{
1828 {
1829 % from bottom
1830 \ifthenelse{\equal{#4}{north east}}{\def\@bbr@tmp@bottomanchor{south east}}{}
1831 \ifthenelse{\equal{#4}{north west}}{\def\@bbr@tmp@bottomanchor{south west}}{}
1832
1833 \ifdefempty{\@bbrcomside}{
1834 \coordinate[#3=-\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.\@bbr@tmp@bottomanchor] (\@bbrcom
1835 ){
1836 \node[#3=-\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.\@bbr@tmp@bottomanchor,anchor=#6,BBRCOM
1837 }
1838 }
1839 {
1840 % from top
1841 \ifdefempty{\@bbrcomside}{
1842 \coordinate[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4] (\@bbrcomsidename);
1843 ){
1844 \node[#3=\@bb@com@tmpoffset and \@bbrcomlength of \@bb@lastbox.#4,anchor=#6,BBRCOM-SIDESTYLE] (\@bbrcom
1845 }
1846 }
1847 \path[#2] (\@bbrcomsidename.#6) edge[BBRCOM-EDGESTYLE] node[above,BBRCOM-TOPSTYLE] (\@bbrcomtopname) {}
1848 %
1849 \@bb@comfinalize{#8}
1850 }
```

\bbrmsgto

\bbrmsgfrom

\bbrmsgtofrom

\bbrmsgfromto

```
1851 \newcommandx{\bbrmsgto}[1]{%
1852 \@bbrmsg{#1}{->}{below left}{north west}{west}{east}{\@bb@message@voffset}{\bbrmsgspace}{\@bbr@first@msg}
1853 }
1854 \newcommandx{\bbrmsgfrom}[1]{%
1855 \@bbrmsg{#1}{<-}{below left}{north west}{west}{east}{\@bb@message@voffset}{\bbrmsgspace}{\@bbr@first@msg}
1856 }
1857
1858 \newcommandx{\bbrmsgtofrom}[2]{%
1859 \bbrmsgto{#1}
1860 \bbrmsgspace{-\@bbr@intermessage@skip}
1861 \bbrmsgspace{\@bbr@intermessage@shortskip}
1862 \bbrmsgfrom{#2}
1863 \bbrmsgspace{\@bbr@intermessage@medskip}
1864 }
1865
1866 \newcommandx{\bbrmsgfromto}[2]{%
1867 \bbrmsgfrom{#1}
1868 \bbrmsgspace{-\@bbr@intermessage@skip}
1869 \bbrmsgspace{\@bbr@intermessage@shortskip}
```

```

1870 \bbrmsgto{#2}
1871 \bbrmsgspace{\@bbr@intermessage@medskip}
1872 }

```

\bbrmsgvdots

```

1873 \newcommand{\bbrmsgvdots}[1] [] {%
1874 \bbrmsgtxt[xshift=\@bbrcomlength/2,afterskip=\@bbr@intermessage@shortskip,#1]{$\vdots$}
1875 }

```

\bbrqryto

```

\bbrqryfrom 1876 \newcommandx{\bbrqryto}[1] {%
\bbrqrytofrom 1877 \@bbrmsg{#1}{<-}{below right}{north east}{east}{west}{\@bb@query@voffset}{\bbrqryspace}{\@bbr@first@qu
\bbrqryfromto 1878 }
1879 \newcommandx{\bbrqryfrom}[1] {%
1880 \@bbrmsg{#1}{->}{below right}{north east}{east}{west}{\@bb@query@voffset}{\bbrqryspace}{\@bbr@first@qu
1881 }
1882
1883 \newcommand*{\bbrqrytofrom}[2] {%
1884 \bbrqryto{#1}
1885 \bbrqryspace{-\@bbr@intermessage@skip}
1886 \bbrqryspace{\@bbr@intermessage@shortskip}
1887 \bbrqryfrom{#2}
1888 \bbrqryspace{\@bbr@intermessage@medskip}
1889 }
1890
1891 \newcommand*{\bbrqryfromto}[2] {%
1892 \bbrqryfrom{#1}
1893 \bbrqryspace{-\@bbr@intermessage@skip}
1894 \bbrqryspace{\@bbr@intermessage@shortskip}
1895 \bbrqryto{#2}
1896 \bbrqryspace{\@bbr@intermessage@medskip}
1897 }

```

\bbrqryvdots

```

1898 \newcommand{\bbrqryvdots}[1] [] {%
1899 \bbrqrytxt[xshift=\@bbrcomlength/2,afterskip=\@bbr@intermessage@skip,#1]{$\vdots$}
1900 }

```

\@bbroracleqry

```

1901 \newcommand{\@bbroracleqry}[4] {
1902 \@bb@comsetup{#1}{#3}{#4}{\@bbr@first@oraclequery}
1903 %
1904 \ifthenelse{\equal{\@bbrcomfixedboffset}{true}}{
1905 {
1906 % from bottom
1907 \path[#2] (\@bb@lastoracle.south west) -- ++ (0,\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0pt,ar
1908 }
1909 {
1910 \path[#2] (\@bb@lastoracle.north west) -- ++ (0,-\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0pt,ar
1911 }
1912 %
1913 \@bb@comfinalize{#4}
1914 }

```

\bbroracleqryto

\bbroracleqryfrom

\bbroracleqrytofrom

\bbroracleqryfromto

```

1915 \newcommand{\bbroracqryfrom}[1]{
1916 \@bbroracqry{#1}{->}{\@bb@oracquery@voffset}{\bbroracqryspace}
1917 }
1918
1919 \newcommand{\bbroracqryto}[1]{
1920 \@bbroracqry{#1}{<-}{\@bb@oracquery@voffset}{\bbroracqryspace}
1921 }
1922
1923 \newcommand*{\bbroracqrytofrom}[2]{%
1924 \bbroracqryto{#1}
1925 \bbroracqryspace{-\@bbr@intermessage@skip}
1926 \bbroracqryspace{\@bbr@intermessage@shortskip}
1927 \bbroracqryfrom{#2}
1928 \bbroracqryspace{\@bbr@intermessage@medskip}
1929 }
1930
1931 \newcommand*{\bbroracqryfromto}[2]{%
1932 \bbroracqryfrom{#1}
1933 \bbroracqryspace{-\@bbr@intermessage@skip}
1934 \bbroracqryspace{\@bbr@intermessage@shortskip}
1935 \bbroracqryto{#2}
1936 \bbroracqryspace{\@bbr@intermessage@medskip}
1937 }

```

\@bbrchallengerqry

```

1938 \newcommand{\@bbrchallengerqry}[4]{
1939 \@bb@comsetup{#1}{#3}{#4}{\@bbr@first@challengerquery}
1940 %
1941 \ifthenelse{equal{\@bbr@comfixedboffset}{true}}{
1942 {
1943 \path[#2] (\@bb@lastchallenger.south east) -- ++ (0,\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0pt]{}
1944 }
1945 {
1946 \path[#2] (\@bb@lastchallenger.north east) -- ++ (0,-\@bb@com@tmpoffset) node[inner sep=0pt,outer sep=0pt]{}
1947 }
1948 %
1949 \@bb@comfinalize{#4}
1950 }

```

\bbroracqryto

```

\bbroracqryfrom 1951 \newcommand{\bbrchallengerqryfrom}[1]{
\bbroracqrytofrom 1952 \@bbrchallengerqry{#1}{<-}{\@bb@challengerquery@voffset}{\bbrchallengerqryspace}
\bbroracqryfromto 1953 }
1954
1955 \newcommand{\bbrchallengerqryto}[1]{
1956 \@bbrchallengerqry{#1}{->}{\@bb@challengerquery@voffset}{\bbrchallengerqryspace}
1957 }
1958
1959 \newcommand*{\bbrchallengerqrytofrom}[2]{%
1960 \bbrchallengerqryto{#1}
1961 \bbrchallengerqryspace{-\@bbr@intermessage@skip}
1962 \bbrchallengerqryspace{\@bbr@intermessage@shortskip}
1963 \bbrchallengerqryfrom{#2}
1964 \bbrchallengerqryspace{\@bbr@intermessage@medskip}
1965 }
1966

```

```

1967 \newcommand*\bbrchallengerqryfromto}[2]{%
1968 \bbrchallengerqryfrom{#1}
1969 \bbrchallengerqryspspace{-\@bbr@intermessage@skip}
1970 \bbrchallengerqryspspace{\@bbr@intermessage@shortskip}
1971 \bbrchallengerqryto{#2}
1972 \bbrchallengerqryspspace{\@bbr@intermessage@medskip}
1973 }

1974
1975
1976 \newcommand*\bbrcomloopleft{}
1977 \newcommand*\bbrcomloopleftstyle{}
1978 \newcommand*\bbrcomloopright{}
1979 \newcommand*\bbrcomlooprightstyle{}
1980 \newcommand*\bbrcomloopcenter{}
1981 \newcommand*\bbrcomloopcenterstyle{}
1982 \newcommand*\bbrcomloopclockwise{false}
1983 \newcommand*\bbrcomloopangle{50}
1984 \define@key{bbrcomloop}{left}[]{\renewcommand*\bbrcomloopleft{#1}}
1985 \define@key{bbrcomloop}{leftstyle}[]{\renewcommand*\bbrcomloopleftstyle{#1}}
1986 \define@key{bbrcomloop}{right}[]{\renewcommand*\bbrcomloopright{#1}}
1987 \define@key{bbrcomloop}{rightstyle}[]{\renewcommand*\bbrcomlooprightstyle{#1}}
1988 \define@key{bbrcomloop}{center}[]{\renewcommand*\bbrcomloopcenter{#1}}
1989 \define@key{bbrcomloop}{centerstyle}[]{\renewcommand*\bbrcomloopcenterstyle{#1}}
1990 \define@key{bbrcomloop}{angle}[]{\renewcommand*\bbrcomloopangle{#1}}
1991 \define@key{bbrcomloop}{clockwise}[true]{\renewcommand*\bbrcomloopclockwise{#1}}
1992
1993 \newcommand{\bbrloop}[3]{
1994 \begingroup % for local keys
1995 \setkeys{bbrcomloop}{#3}%
1996
1997 \tikzset{BBRLOOP-LEFTSTYLE/.style/.expand once=\bbrcomloopleftstyle}%
1998 \tikzset{BBRLOOP-RIGHTSTYLE/.style/.expand once=\bbrcomlooprightstyle}%
1999 \tikzset{BBRLOOP-CENTERSTYLE/.style/.expand once=\bbrcomloopcenterstyle}%
2000
2001
2002 \ifthenelse{\equal{\bbrcomloopclockwise}{true}}{
2003 {
2004 \path[->] (#1) edge[bend left=\bbrcomloopangle] node[midway,left,inner sep=0,outer sep=0,BBRLOOP-LEFTSTYLE]{}
2005 \path[->] (#2) edge[bend left=\bbrcomloopangle] node[midway,right,inner sep=0,outer sep=0,BBRLOOP-RIGHTSTYLE]{}
2006 }
2007 {
2008 \path[->] (#1) edge[bend right=\bbrcomloopangle] node[midway,left,inner sep=0,outer sep=0,] (bbrleft) {}
2009 \path[->] (#2) edge[bend right=\bbrcomloopangle] node[midway,right,inner sep=0,outer sep=0,] (bbrright) {}
2010 }
2011 \node[at=($ (bbrleft.west)!0.5!(bbrright.east)$),anchor=center,BBRLOOP-CENTERSTYLE]{} {\bbrcomloopcenter}
2012
2013 \endgroup
2014 }
2015
2016 \newcommand*\bbrintertextthoffset{1.5cm}
2017 \define@key{bbrintertext}{xshift}[]{\renewcommand*\bbrintertextthoffset{#1}}
2018
2019 \newcommand{\@bb@intertextsetup}[1]{
2020 %load keys
2021 \begingroup % for local keys

```



```

2022
2023 % fix align environment (e.g. for use of pseudocode)
2024 % ^^A https://tex.stackexchange.com/questions/36954/spurious-space-above-align-environment-at-top-of-p
2025 %\pretocmd\start@align{%
2026   %\if@minipage\kern-0.5\abovedisplayskip\fi
2027 %}\{}{}
2028
2029 \setkeys{bbrcom,bbrabase,bbrintertext}{#1}%
2030 \@bbrcom@check@islast{}
2031
2032 \tikzset{BBRBASE-NODESTYLE/.style/.expand once=\@bbrbasenodestyle}%
2033 }
2034
2035 \newcommand{\@bb@intertextfinalize}[1]{
2036 #1{\@bbrcomafterskip}
2037 \endgroup
2038 \def\@bbr@lastskip{\@bbr@intermessage@veryshortskip}
2039 }

\@bbrintertext 7 -> whether or not this is the first msg/query
2040 \newcommand{\@bbrintertext}[7]{
2041 \edef\@tmp@bbr@isfirst{#7}
2042 \renewcommand#7{false}
2043
2044 \@bb@intertextsetup{#1}
2045
2046 % increase space
2047 #5{\@bbrcombeforeskip}
2048 \ifthenelse{\equal{\@bbrcomfixedoffset}{}}{
2049 {
2050 \ifthenelse{\equal{\@tmp@bbr@isfirst}{true}}{
2051 }{#5{\@bbr@intermessage@veryshortskip}}
2052
2053 \setlength{\@bb@com@tmpoffset}{#4}%
2054 }
2055 {
2056 \setlength{\@bb@com@tmpoffset}{\@bbrcomfixedoffset}%
2057 }
2058
2059 %
2060 \ifthenelse{\equal{\@bbrcomfixedboffset}{true}}{
2061 {
2062 % from bottom
2063 \ifthenelse{\equal{#3}{north east}}{\def\@bbr@tmp@bottomanchor{south east}}{}
2064 \ifthenelse{\equal{#3}{north west}}{\def\@bbr@tmp@bottomanchor{south west}}{}
2065
2066 \node[#2=-\@bb@com@tmpoffset and \bbrintertextxoffset of \@bb@lastbox.\@bbr@tmp@bottomanchor, inner sep=
2067 }
2068 {
2069 \node[#2=\@bb@com@tmpoffset and \bbrintertextxoffset of \@bb@lastbox.#3, inner sep=0, outer sep=0, BBR
2070 }
2071 %
2072 % compute height of node
2073 \coordinate (@btmpcoord) at (\@bbrbasenodename.north);
2074 \path (@btmpcoord);
2075 \pgfgetlastxy{\XCoord}{\YCoordA}

```

```

2076 \coordinate (@bbtmpcoord) at (\@bbrbasenodename.south);
2077 \path (@bbtmpcoord);
2078 \pgfgetlastxy{\XCoord}{\YCoordB}
2079
2080 % update voffset
2081 \setlength{\@bb@tmplength@b}{\YCoordA-\YCoordB}
2082 #5{\the\@bb@tmplength@b}
2083
2084 \@bb@intertextfinalize{#5}
2085 }

2086 \newcommand{\bbrmsgtxt}[2][ ]{
2087 \@bbrintertext{#1}{below left}{north west}{\@bb@message@voffset}{\bbrmsgspace}{#2}{\@bbr@first@msg}
2088 }
2089
2090 \newcommand{\bbrqrytxt}[2][ ]{
2091 \@bbrintertext{#1}{below right}{north east}{\@bb@query@voffset}{\bbrqryspace}{#2}{\@bbr@first@query}
2092 }
2093
2094 \newcommand{\bbrchallengertxt}[2][ ]{
2095 \begingroup
2096 \setlength{\@bb@tmplength@b}{\bbrchallengerhdistance/2}%
2097 \renewcommand{\bbrintertext@offset}{\the\@bb@tmplength@b}%
2098 \@bbrintertext{#1}{below left}{north west}{\@bb@challengequery@voffset}{\bbrchallengerqryspace}{#2}{\@bbr@first@challenge}
2099 \endgroup
2100 }
2101
2102 \newcommand{\bbroracletxt}[2][ ]{
2103 \begingroup
2104 \setlength{\@bb@tmplength@b}{\bbroraclehdistance/2}%
2105 \renewcommand{\bbrintertext@offset}{\the\@bb@tmplength@b}%
2106 \@bbrintertext{#1}{below left}{north west}{\@bb@oraclequery@voffset}{\bbroracleqryspace}{#2}{\@bbr@first@oracle}
2107 \endgroup
2108 }
2109
2110 \newcommand{\bbrmsgspace}[1]{
2111 \@pc@globaladdtolength{\@bb@message@voffset}{#1}
2112 }
2113
2114 \newcommand{\bbrqryspace}[1]{
2115 \@pc@globaladdtolength{\@bb@query@voffset}{#1}
2116 }
2117
2118 \newcommand{\bbroracleqryspace}[1]{
2119 \@pc@globaladdtolength{\@bb@oraclequery@voffset}{#1}
2120 }
2121
2122 \newcommand{\bbrchallengerqryspace}[1]{
2123 \@pc@globaladdtolength{\@bb@challengequery@voffset}{#1}
2124 }
2125
2126

```

9.12 Game-Based Proofs

2127

`gamechange` Highlighting of changes between games. Highlight color can be set via `\gamechange color`

`\pobox` A simple box for conditional (ie., boxed) lines.

\pcgame

```
2141 \def\pcgame{\bgroup\pcgame@}
```

```
2143 \def\pcgame@@{\pcgameprocedurestyle{\pcgamename}\egroup}
```

`\@pc@gametitle` Creates the header/title of a game

\gameprocedurearg

gameproof

2150 \newcommand*\@pcgameproofgamenr{0}

```
2151 \define@key{pcgameproof}{nr}[]{\renewcommand*\@pcgameproofgamenr{#1}}
```

```
2152 \define@key{pcgameproof}{name}[]{\renewcommand*{pcgamenname{\ensuremath{#1}}}}
```

```
2153 \define@key{pcgameproof}{arg}[]{\renewcommand*\gameprocedurearg{\ensuremath{#1}}}
```

2154

```
2155 \newenvironment{gameproof}[1] [] {%
```

2156 \begingroup%

```
2157 \setkeys{pcgameproof}{#1}%
```

2158 \@pc@ensureremember%

2159 \setcounter{pcgamecounter}{\@pcgameproofgamenr}%

```
2160 \setcounter{pcstartgamecounter}{\@pcgameproofgamenr}\stepcounter{pcstartgamecounter}%
```

2161 }{\@pc@releaseremember\endgroup}

```
2162 \newcommand{\setgameproceduredefaultstyle}[1]{%
```

```
2163 \PackageWarning{cryptocode}{Deprecated command setgameproceduredefaultstyle. Use pcsetargs instead.}%
```

```
2164 \pcsetargs{#1}}
```

2165

```
2166 \createpseudocodecommand{gameprocedure}
```

```
2167 {\addtocounter{pcgamecounter}{1}\renewcommand{\@withingame}{true}}
```

```
2168    {\@pc@gametitle}
```

2169 $\{ \}$

2170

```

2171 \def\@bxgame@pseudocodeA[#1]#2#3{\setkeys*{pcspace}{#1}\renewcommand{\@bxgameheader}{\@pc@gametitle[#2]
2172 \@pseudocode[head=\@pc@gametitle,#1]{#3}}
2173 \def\@bxgame@pseudocodeB#1#2{\renewcommand{\@bxgameheader}{\@pc@gametitle[#1]}}%
2174 \@pseudocode[head=\@pc@gametitle]{#2}}
2175
2176 \newcommand{\bxgameprocedure}{
2177 \begingroup%
2178 \renewcommand{\@withinspaces}{false}%
2179 \renewcommand{\@withingame}{true}%
2180 \renewcommand{\@withinbxgame}{true}%
2181 \stepcounter{pcgamecounter}%
2182 \@ifnextchar[%]
2183   {\@bxgame@pseudocodeA}
2184   {\@bxgame@pseudocodeB}%
2185 }
2186
2187 \newcommand{\@pc@secondheader}{}
2188
2189 %tbx top boxed
2190 \createpseudocodecommand{tbxgameprocedure}
2191 {\addtocounter{pcgamecounter}{1}\renewcommand{\@withingame}{true}%
2192 \renewcommand{\@pc@secondheader}{true}}
2193 {\@pc@gametitle}
2194 {}
2195
2196
2197 \newcommand*\@pcgamehopnodestyle{}
2198 \newcommand*\@pcgamehopedgestyle{bend left}
2199 \newcommand*\@pcgamehoppathestyle{}
2200 \newcommand*\@pcgamehophint{}
2201 \newcommand*\@pcgamehophintbelow{}
2202 \newcommand*\@pcgamehopinhint{}
2203 \newcommand*\@pcgamehoplength{1.5cm}
2204 \define@key{pcgamehop}{nodestyle}[]{\renewcommand*\@pcgamehopnodestyle{#1}}
2205 \define@key{pcgamehop}{edgestyle}[]{\renewcommand*\@pcgamehopedgestyle{#1}}
2206 \define@key{pcgamehop}{pathstyle}[]{\renewcommand*\@pcgamehoppathestyle{#1}}
2207 \define@key{pcgamehop}{hint}[]{\renewcommand*\@pcgamehophint{#1}}
2208 \define@key{pcgamehop}{belowhint}[]{\renewcommand*\@pcgamehophintbelow{#1}}
2209 \define@key{pcgamehop}{inhint}[]{\renewcommand*\@pcgamehopinhint{#1}}
2210 \define@key{pcgamehop}{length}[]{\renewcommand*\@pcgamehoplength{#1}}
2211
2212
2213 \newcommand{\@pc@setupgamehop}[1]{
2214 \begingroup\setkeys{pcgamehop}{#1}%
2215 \tikzset{GAMEHOP-PATH-STYLE/.style/.expand once=\@pcgamehoppathestyle}%
2216 \tikzset{GAMEHOP-NODE-STYLE/.style/.expand once=\@pcgamehopnodestyle}%
2217 \tikzset{GAMEHOP-EDGE-STYLE/.style/.expand once=\@pcgamehopedgestyle}%
2218 }
2219
2220 \newcommand{\@pc@finalizgamehop}{
2221 \endgroup
2222 }
2223
2224 \newcommandx*\@addgamehop}[3]{%
2225 \begingroup%
2226 \ifthenelse{#1<#2}%

```

```

2227 {\ifthenelse{\equal{\@withingamedescription}{true}}{%
2228 {\renewcommand*\@pcgamehopedgestyle{bend right=20}\renewcommand*\@pcgamehopnodestyle{rotate=90}}{}}%
2229 }%
2230 {\renewcommand*\@pcgamehopedgestyle{bend right}}%
2231 \@pc@setupgamehop{#3}%
2232 \begin{tikzpicture}[overlay]%
2233 \ifthenelse{#1<#2}{%
2234 \path[->,GAMEHOP-PATH-STYLE] (gamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE]
2235 node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (gamenode#2);
2236 }{%
2237 \path[->,GAMEHOP-PATH-STYLE] (bgamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE]
2238 node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (bgamenode#2);
2239 }%
2240 \end{tikzpicture}%
2241 \@pc@finalizegamehop%
2242 \endgroup%
2243 }
2244 \newcommandx*\@addstartgamehop}[2][1=\thepcstartgamecounter]{%
2245 \@pc@setupgamehop{#2}
2246 \begin{tikzpicture}[overlay]
2247 \node[left=\@pcgamehoplength of gamenode#1] (tmpgamenode0) {};
2248 \path[->,GAMEHOP-PATH-STYLE] (tmpgamenode0) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE]
2249 node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (gamenode#1);
2250 \end{tikzpicture}
2251 \@pc@finalizegamehop
2252 }
2253 \newcommandx*\@addendgamehop}[2][1=\thepcgamecounter]{%
2254 \@pc@setupgamehop{#2}
2255 \begin{tikzpicture}[overlay]
2256 \node[right=\@pcgamehoplength of gamenode#1] (tmpgamenode#1) {};
2257 \path[->,GAMEHOP-PATH-STYLE] (gamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE]
2258 node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (tmpgamenode#1);
2259 \end{tikzpicture}
2260 \@pc@finalizegamehop
2261 }
2262 \newcommandx*\@addbxgamehop}[3]{%
2263 \@pc@setupgamehop{#3}
2264 \begin{tikzpicture}[overlay]
2265 \path[->,GAMEHOP-PATH-STYLE] (bgamenode#1) edge[GAMEHOP-EDGE-STYLE] node[above,GAMEHOP-NODE-STYLE]
2266 node[below,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (bgamenode#2);
2267 \end{tikzpicture}
2268 \@pc@finalizegamehop
2269 }
2270 \newcommandx*\@addloopgamehop}[2][1=\thepcgamecounter]{%
2271 \@pc@setupgamehop{#2}
2272 \begin{tikzpicture}[overlay]
2273 \node (looptemp1) [right=0.5cm of gamenode#1] {};
2274 \draw[->,GAMEHOP-PATH-STYLE] (gamenode#1) -- (looptemp1|-gamenode#1) -- node[right,GAMEHOP-NODE-STYLE]
2275 node[left,GAMEHOP-NODE-STYLE] {\@pcgamehophintbelow} (looptemp1|-bgamenode#1)-- (bgamenode#1);
2276 \end{tikzpicture}
2277 \@pc@finalizegamehop
2278 }
2279
2280

```

9.12.1 Game Descriptions

```

2281
2282 \newenvironment{gamedescription}[1][1]{%
2283 \begin{group}%
2284 \setkeys{pcgameproof}{#1}%
2285 \renewcommand{\@withingamedescription}{true}%
2286 \@pc@ensureremember%
2287 \setcounter{pcgamecounter}{\@pcgameproofgamenr}%
2288 \setcounter{pcstartgamecounter}{\@pcgameproofgamenr}\stepcounter{pcstartgamecounter}%
2289 \begin{description}%
2290 }\end{description}\@pc@releaseremember\endgroup}
2291
2292 \newcommandx*{\describegame}[1][1=]{%
2293 \addtocounter{pcgamecounter}{1}%
2294 \item[%
2295 \pcdraw{
2296 \gdef\i{\thepcgamecounter}%
2297 \node[inner sep=0.0em,outer sep=0, xshift=-1ex, yshift=0.5ex] (gamenode\i) {};
2298 }%
2299 \@pc@gametitle:]}%
2300 \begin{group}\setkeys{pcgamehop}{#1}%
2301 \ifthenelse{\equal{\@pcgamehophint}}{
2302 {}
2303 {\hspace{-0.7ex}\pcdraw{%the -0.7ex is a horrible hack to fix a whitespace issue with tikz (see http
2304 \tikzset{GAMEHOP-PATH-STYLE/.style/.expand once=\@pcgamehoppathstyle}%
2305 \tikzset{GAMEHOP-NODE-STYLE/.style/.expand once=\@pcgamehopnodestyle}%
2306 \draw[->,GAMEHOP-PATH-STYLE] (gamenode\thepcgamecounter) --++ (0,-\@pcgamehoplength) node[midway,above,
2307 ]}%
2308 \ifthenelse{\equal{\@pcgamehopinhint}}{
2309 {}
2310 {\hspace{-0.7ex}\pcdraw{%the -0.7ex is a horrible hack to fix a whitespace issue with tikz (see http
2311 \tikzset{GAMEHOP-PATH-STYLE/.style/.expand once=\@pcgamehoppathstyle}%
2312 \tikzset{GAMEHOP-NODE-STYLE/.style/.expand once=\@pcgamehopnodestyle}%
2313 \draw[<-,GAMEHOP-PATH-STYLE] (gamenode\thepcgamecounter) --++ (0,\@pcgamehoplength) node[midway,above,
2314 ]}%
2315 }%
2316 \endgroup}%
2317 }
2318 %
2319 % \end{macrocode}
2320 %
2321 %
2322 % \iffalse
2323 % \begin{macrocode}
2324 % \fi
2325 \end{cryptocode.sty}

```

Change History

v0.04

General: added \pcabort. 1
better control whitespace for \pcif,
\pcelse, \pcelseif. 1

v0.05

General: add bottom to namepos in
bbrbox 1
angle for bbrloop 1

fix length for bbrinput	1	Added <code>\pindist</code> , <code>\sindist</code> , and <code>\cindist</code> to operators.	1
introduce hoffset for bbrinput	1	Added <code>aboveskip</code> and <code>belowskip</code> option to <code>\pchstack</code> and <code>\pcvstack</code>	1
names for bbrinput and bbroutput	1	Added additional adversaries.	1
side and oside support to <code>\bbroracleqcrypto</code> and <code>\bbroracleqcryptofrom</code>	1	Added additional complexity classes.	1
v0.06		Added additional polynomials.	1
General: added <code>\pcunless</code>	1	Added block forms for pseudocode and procedure commands (<code>\pseudocodeblock</code> and <code>\procedureblock</code>).	1
v0.10		Added boxed, inline, noindent options to <code>\pchstack</code> and <code>\pcvstack</code>	1
General: Initial version	1	Added clockwise, leftstyle, centerstyle, rightstyle for <code>bbrloop</code> . Adjusted placing of center.	1
v0.11		Added command <code>\pcsetargs</code> to define default arguments for pseudocode blocks.	1
General: Added <code>pcmbbox</code> environment for matrices in pseudocode.	1	Added command <code>\pcsethstackargs</code> and <code>\pcsetvstackargs</code> to define default arguments for <code>hstack</code> and <code>vstack</code> environments.	1
Added <code>\NAND</code> command.	1	Added <code>fixedoffset</code> , <code>fixedboffset</code> , <code>islast</code> for reduction messages.	1
changed command <code>pckeystyle</code> to ensure that subscripts on <code>sk</code> and <code>pk</code> are aligned the same before, (<code>sk_R</code> , <code>pk_R</code>) had slightly misaligned subscripts due to Tex treating subscripts on composite objects with descenders differently than without.	1	Added <code>headheight</code> option to <code>\pseudocode</code>	1
v0.20		Added <code>minlineheight</code> option to <code>\pseudocode</code>	1
General: Added <code>\pcfail</code>	1	Added <code>oracles</code> package option.	1
Added <code>namepos</code> middle for <code>bbrbox</code>	1	Added <code>space</code> option to <code>\pchstack</code> and <code>\pcvstack</code>	1
Added <code>valign</code> to pseudocode to allow minipage vertical alignment.	1	Adjusted spacing via <code>\pcaboveskip</code> and <code>\pcbelowskip</code> which are added to <code>\pseudocode</code> blocks and <code>pchstack</code> environments	1
Changed <code>minheight</code> for <code>bbrbox</code> environment to actually reflect a minimum height in <code>tikz</code> . The old <code>minheight</code> which added space at the bottom was preserved as <code>addheight</code>	1	Bigger refactoring. Not completely backwards compatible. In particular, optimized spacing of pseudocode blocks and black box reductions.	1
Ensure line numbers are right aligned to allow for two digit linenumbers having the same width.	1	Fixed spacing issues with black box reduction messages.	1
v0.30		Renamed horizontal spacing commands <code>\beforepcskip</code> and <code>\afterpcskip</code> to <code>\pcbeforeskip</code> and <code>\pcafterskip</code>	1
General: replace obsolete <code>l3regex</code>	1	Switched to <code>mathtools</code> <code>DeclarePairedDelimiter</code> for paired operators. Each paired operator comes in two forms, e.g, <code>abs</code> and <code>tabs</code> the latter to be used in flowtext which does not scale the outer delimiters.	1
v0.31			
General: added <code>\prp</code>	1		
added <code>\tprob</code> (variants for <code>prob</code> and <code>co</code> for in-text)	1		
v0.32			
General: allow overwriting rule command in pseudocode via <code>headlinecmd</code> (defaults to <code>\hrule</code>)	1		
allow to control spacing with <code>\pcfor</code>	1		
v0.40			
General: Adapted <code>bbenv</code> environment to take key value option list. Old format is still supported but deprecated.	1		
Added <code>\argmax</code> and <code>\argmin</code> to operators.	1		

v0.41		v0.42	
<code>\bbrqrvydots</code> : Added <code>bbrqrvydots</code>	114	General: Added command <code>\pcassert</code>	1
General: Fixed horizontal spacing in <code>bbrenv</code> environment.	1	More robust <code>\sample</code> command that also works in subscripts.	1
<code>bbrenv</code> : Added <code>tikzargs</code> key to pass in arguments to surrounding <code>tikzpicture</code>	106	v0.43	
Fixed horizontal spacing behind <code>bbrenv</code> blocks.	106	General: Added support for <code>cleveref</code> via <code>\pcfixcleveref</code> command.	1
		v0.44	
		General: Removed dependency on deprecated <code>etex</code> package.	1