User's Guide for complexity: a LaTeX package, Version 0.81a

Chris Bourke

September 1, 2018

Contents

1 Introduction

1.1 What is complexity?

complexity is a LaTeX package that typesets computational complexity classes such as P (deterministic polynomial time) and NP (nondeterministic polynomial time) as well as sets (languages) such as SAT (satisfiability). In all, over 350 commands are defined for helping you to typeset Computational Complexity constructs.

2 Package Options

The complexity package provides two general options—a *font* option (of which there are three classes) and a *mode* option. The font option specifies what font the complexity classes (as well as functions and languages) are typeset in while the mode option specifies *how many* complexity classes are defined.

One specifies these options in the usual manner. When you use the package, you can pass it the options you wish; for example, calling the package with

```
\usepackage[bold,full]{complexity}
```

specifies that classes (and languages) should be typeset in bold and that the full list of classes should be defined. Invalid options are ignored and only the last option (of each type) is used if multiple, conflicting options are given. The complete options are described in the next two subsections.

2.1 Mode Options

The mode options specify to what extent the package declares commands for complexity classes. By default, *every* (supported) class command is defined. Alternatively, you can limit the number of commands the `complexity` package defines (and perhaps limit conflicts with other packages or your own commands) by using the `basic` option. This option defines only the most commonly used complexity classes.

`full` (*Default*) This option will load *every* complexity class that the package has defined. See Section **??** for a complete list.

`basic` This option will only load the "standard" complexity classes so as to minimize the number of commands the package defines (i.e. standard classes like P and NP but not less well known classes like AWPP (Almost wide PP).

2.2 Font Options

You can easily change the fonts for complexity classes using a package option. The `complexity` package defines three different font entities: a font for complexity classes (`classfont`), a font for languages (`langfont`), and a font for functions (`funcfont`). By default, all of these fonts are typeset using the `mathsf` font. You can change the font for all of them together or specify a font for each individually. To apply a single font to all three entities, simply pass the font (by itself) as an option. The supported font options are as follows.

`sanserif` (*Default*) This typesets the classes in a `\mathsf` (sans serif) font.

`roman` This option typesets the classes in a `\mathrm` (roman) font.

`bold` This option typesets the classes in a `\mathbf` (roman, bold) font.

`typewriter` This option typesets the classes in a `\mathtt` (typewriter) font.

`italic` This option typesets the classes in a `\mathit` (math italic) font.

`caps` This option typesets the classes in a `\textsc` (small caps font) font.

`slant` This option typesets the classes in a `\textsl` (slanted font) font.

As an alternative, you can specify a different font for each of the three entities. To do this, you simply qualify the font with a key-value pair: either `classfont`, `langfont`, or `funcfont`. For example, if we want our complexity classes to be typeset in `bold`, our languages to be typeset in `roman` and our functions to be typeset in `italic`, we would call the package using:

```
\usepackage[classfont=bold,
            langfont=roman,
            funcfont=italic]{complexity}
```

Examples of how each of the fonts appears when typeset can be found in Table ??.

Table 1: An Example of each font

Font	`classfont`	`langfont`	`funcfont`
`sanserif`	$\mathsf{P} \subseteq \mathsf{NP}$, $\mathsf{PSPACE} \subseteq \mathsf{EXP}$	$\mathsf{CVP} \leq_m \mathsf{SAT}$, $\mathsf{SAT} \leq_T \mathsf{MaxSAT}$	$\mathsf{polylog} \in O(\mathsf{poly})$, $\mathsf{polylog} \in \Omega(\mathsf{log})$
`roman`	$\mathrm{P} \subseteq \mathrm{NP}$, $\mathrm{PSPACE} \subseteq \mathrm{EXP}$	$\mathrm{CVP} \leq_m \mathrm{SAT}$, $\mathrm{SAT} \leq_T \mathrm{MaxSAT}$	$\mathrm{polylog} \in O(\mathrm{poly})$, $\mathrm{polylog} \in \Omega(\mathrm{log})$
`bold`	$\mathbf{P} \subseteq \mathbf{NP}$, $\mathbf{PSPACE} \subseteq \mathbf{EXP}$	$\mathbf{CVP} \leq_m \mathbf{SAT}$, $\mathbf{SAT} \leq_T \mathbf{MaxSAT}$	$\mathbf{polylog} \in O(\mathbf{poly})$, $\mathbf{polylog} \in \Omega(\mathbf{log})$
`typewriter`	$\mathtt{P} \subseteq \mathtt{NP}$, $\mathtt{PSPACE} \subseteq \mathtt{EXP}$	$\mathtt{CVP} \leq_m \mathtt{SAT}$, $\mathtt{SAT} \leq_T \mathtt{MaxSAT}$	$\mathtt{polylog} \in O(\mathtt{poly})$, $\mathtt{polylog} \in \Omega(\mathtt{log})$
`italic`	$\mathit{P} \subseteq \mathit{NP}$, $\mathit{PSPACE} \subseteq \mathit{EXP}$	$\mathit{CVP} \leq_m \mathit{SAT}$, $\mathit{SAT} \leq_T \mathit{MaxSAT}$	$\mathit{polylog} \in O(\mathit{poly})$, $\mathit{polylog} \in \Omega(\mathit{log})$
`caps`	$\mathrm{P} \subseteq \mathrm{NP}$, $\mathrm{PSPACE} \subseteq \mathrm{EXP}$ Better example: $\textsc{promise}\mathrm{RP} \subseteq \textsc{promise}\mathrm{BPP}$	$\mathrm{CVP} \leq_m \mathrm{SAT}$, $\mathrm{SAT} \leq_T \textsc{MaxSAT}$	$\textsc{polylog} \in O(\textsc{poly})$, $\textsc{polylog} \in \Omega(\textsc{log})$
`slant`	$\mathit{P} \subseteq \mathit{NP}$, $\mathit{PSPACE} \subseteq \mathit{EXP}$	$\mathit{CVP} \leq_m \mathit{SAT}$, $\mathit{SAT} \leq_T \mathit{MaxSAT}$	$\mathit{polylog} \in O(\mathit{poly})$, $\mathit{polylog} \in \Omega(\mathit{log})$

2.2.1 The `small` Option

A special option is the `small` option and pertains only to how complexity classes (`classfont`) are typeset. Since classes are typeset in uppercase letters, they tend to be more dominant. If you frequently typeset classes such as PSPACE or DTIME it can interrupt the normal flow of text layouts. One solution to this is to typeset classes 1pt smaller than the surrounding text. This is the approach taken in some texts (most notably, Papadimitriou's book *Computational Complexity*, 1994) and it works quite well. The following samples illustrate the difference. The first sample uses the default font size while the second uses a font that is 1pt smaller (internally, the `\small` command is used). The difference is subtle but when used in a long text, flows more naturally.

> There are deterministic classes such as PSPACE, nondeterministic classes such as NP, and functional classes such as GapP. But I like them all.

> There are deterministic classes such as PSPACE, nondeterministic classes such as NP, and functional classes such as GapP. But I like them all.

To get the same effect using `complexity`, use the `small` option:

```
\usepackage[small]{complexity}
```

with any combination of the other options (it works for all fonts, but some do not look as good as others; `typewriter` for example looks bad with this option). Remember, however that this option only affects how classes are typeset, not languages.

This option only affects how classes are typeset in the display and in-line math-modes. It has no effect in a footnote or some special environment. Subscripts, superscripts (as well as subsubscripts and supersuperscripts) are not effected either. TEX is allowed to automatically change font sizes for these cases.

2.3 Overridden Commands

Three commands in the `complexity` package override built-in TEX commands. Specifically, `\L` (which typesets the symbol Ł), `\P` (typesetting ¶), and `\S` (which typesets the symbol §) are all redefined for use in the package. The `complexity` package preserves these commands so that you may still use them. To use any of these symbols, use the commands `\defaultL`, `\defaultP`, and `\defaultS` instead.

In some situations this redefinition is not desirable (if you need these symbols but cannot use the `\default` versions). You can disable the overriding of these three commands using the option `disableredefinitions`; for example:

```
\usepackage[disableredefinitions]{complexity}
```

Three alternatives are defined if you disable the overrides: `\cL`, `\cP` and `\cS` that typeset the classes L, P and S respectively.

3 Using the Package

Each of the commands is defined using `\ensuremath` so that you need not be in LaTeX's mathmode to use them. However, if you use a command outside of mathmode, TeX may not properly insert surrounding whitespace. It is recommended to always use `complexity` commands inside mathmode. A complete list of commands for classes can be found in Section **??**.

3.1 Special Commands

In addition to complexity classes, the `complexity` package also conveniently defines several commands for commonly used functions and languages. In particular, `\co` (ex: co) and `\parity` (an alias for `\oplus`, typesetting $\oplus$) can be placed preceding a class to refer to the complement or counting versions respectively.

3.2 Function Commands

`complexity` defines several general classes of functions such as logarithms and polynomials. Table **??** gives a complete list of these functions.

Table 2: `func` Commands

Command	Result	Comment
`\llog`	log	Denotes logarithmic functions. The command is invoked with *two* l's so as to not interfere with the LaTeX `\log` command.
`\poly`	poly	Denotes polynomial functions
`\polylog`	polylog	Denotes polylogarithmic functions
`\qpoly`	qpoly	Denotes polynomial functions for quantum advice
`\qlog`	qlog	Denotes logarithmic functions for quantum advice
`\MOD`	MOD	Used for Modular classes/functions
`\Mod`	Mod	Used for Modular classes/functions

3.3 Language Commands

`complexity` also defines commands to typeset languages. A complete list of predefined language commands can be found in Table **??**. The number of commands is sparse; this was intentional. How one refers to languages is far less standard than how one refers to classes. Some people like to explicitly write *every* word (WeightedHamiltonianCycle, or WEIGHTED HAMILTONIAN CYCLE), while others have their own abbreviations. Keeping the number of languages `complexity` defines to a minimum allows for the maximum flexibility.

Table 3: Special `complexity` Commands

Command	Result	Comment
`\CVP`	CVP	Used for the Circuit Value Problem (a P-complete set)
`\SAT`	SAT	Used for Satisfiability (an NP-complete set)
`\MaxSAT`	MaxSAT	Used for the Lexicographically maximum satisfiability optimization problem (complete for OptP)

3.4 Complete List of Class Commands

A complete list (in alpha-numeric order according to the command name) of complexity commands is given below. The first item in each row is the command itself. The second is an example of how it is typeset using the default `sanserif` font. Finally, the third item indicates which mode the command is defined in.

`\AC`	AC	basic
`\A`	A	full
`\ACC`	ACC	basic
`\AH`	AH	basic
`\AL`	AL	basic
`\AlgP`	AlgP	full
`\AM`	AM	basic
`\AMEXP`	AM-EXP	basic
`\Amp`	Amp	full
`\AmpMP`	AmpMP	full
`\AmpPBQP`	AmpPBQP	full
`\AP`	AP	basic
`\APP`	APP	full
`\APX`	APX	full
`\AUCSPACE`	AUC-SPACE	full
`\AuxPDA`	AuxPDA	full
`\AVBPP`	AVBPP	full
`\AvE`	AvE	full
`\AvP`	AvP	full
`\AW`	AW	full
`\AWPP`	AWPP	full
`\betaP`	βP	full
`\BH`	BH	basic
`\BP`	BP	full
`\BPE`	BPE	basic
`\BPEE`	BPEE	basic
`\BPHSPACE`	$\mathsf{BP_HSPACE}$	full

\BPL	BPL	full
\BPP	BPP	basic
\BPPOBDD	BPP-OBDD	full
\BPPpath	$\mathsf{BPP}_{\mathsf{path}}$	full
\BPQP	BPQP	full
\BPSPACE	BPSPACE	basic
\BPTIME	BPTIME	basic
\BQNC	BQNC	full
\BQNP	BQNP	full
\BQP	BQP	basic
\BQPOBDD	BQP-OBDD	full
\BQTIME	BQTIME	basic
\C	C	basic
\cc	cc	basic
\CeL	$\mathsf{C}_=\mathsf{L}$	basic
\CeP	$\mathsf{C}_=\mathsf{P}$	basic
\CFL	CFL	basic
\CH	CH	basic
\CkP	$\mathsf{C}_k\mathsf{P}$	basic
\CLOG	CLOG	full
\CNP	CNP	full
\coAM	coAM	basic
\coBPP	coBPP	basic
\coCeP	$\mathsf{coC}_=\mathsf{P}$	basic
\cofrIP	cofrIP	full
\Coh	Coh	full
\coMA	coMA	basic
\compIP	compIP	full
\compNP	compNP	full
\coNE	coNE	basic
\coNEXP	coNEXP	basic
\coNL	coNL	basic
\coNP	coNP	basic
\coNQP	coNQP	basic
\coRE	coRE	basic
\coRNC	coRNC	basic
\coRP	coRP	basic
\coSL	coSL	basic
\coUCC	coUCC	full
\coUP	coUP	basic
\CP	CP	full
\CSIZE	CSIZE	basic
\CSL	CSL	full
\CZK	CZK	full
\D	D	full
\DCFL	DCFL	full

\DET	DET	basic
\DiffAC	DiffAC	full
\DisNP	DisNP	full
\DistNP	DistNP	full
\DP	DP	full
\DQP	DQP	full
\DSPACE	DSPACE	basic
\DTIME	DTIME	basic
\DTISP	DTISP	basic
\Dyn	Dyn	full
\DynFO	Dyn-FO	full
\E	E	basic
\EE	EE	basic
\EEE	EEE	basic
\EESPACE	EESPACE	basic
\EEXP	EEXP	basic
\EH	EH	basic
\EL	EL	full
\ELEMENTARY	ELEMENTARY	full
\ELkP	$\mathsf{EL}_k\mathsf{P}$	full
\EPTAS	EPTAS	basic
\EQBP	EQBP	full
\EQP	EQP	full
\EQTIME	EQTIME	full
\ESPACE	ESPACE	basic
\ExistsBPP	ExistsBPP	full
\ExistsNISZK	ExistsNISZK	full
\EXP	EXP	basic
\EXPSPACE	EXPSPACE	basic
\FBQP	FBQP	full
\Few	Few	full
\FewP	FewP	full
\FH	FH	full
\FNL	FNL	basic
\FNP	FNP	basic
\FO	FO	full
\FOLL	FOLL	full
\FP	FP	basic
\FPR	FPR	full
\FPRAS	FPRAS	basic
\FPT	FPT	full
\FPTAS	FPTAS	full
\FPTnu	$\mathsf{FPT}_{\mathsf{nu}}$	full
\FPTsu	$\mathsf{FPT}_{\mathsf{su}}$	full
\FQMA	FQMA	basic
\frIP	frIP	full

\FTAPE	F-TAPE	full
\FTIME	F-TIME	full
\G	G	full
\GA	GA	basic
\GANSPACE	GAN-SPACE	full
\Gap	Gap	basic
\GapAC	GapAC	basic
\GapL	GapL	basic
\GapP	GapP	basic
\GC	GC	full
\GCSL	GCSL	full
\GI	GI	basic
\GPCD	GPCD	full
\Heur	Heur	basic
\HeurBPP	HeurBPP	basic
\HeurBPTIME	HeurBPTIME	basic
\HkP	$\mathsf{H}_k\mathsf{P}$	full
\HSPACE	HSPACE	basic
\HVSZK	HVSZK	full
\IC	IC	full
\IP	IP	basic
\IPP	IPP	full
\K	K	basic
\kBQBP	k-BQBP	full
\kBWBP	k-BWBP	full
\kEQBP	k-EQBP	full
\kPBP	k-PBP	full
\KT	KT	basic
\L	L	basic
\LIN	LIN	basic
\LkP	$\mathsf{L}_k\mathsf{P}$	full
\LOGCFL	LOGCFL	full
\LogFew	LogFew	basic
\LogFewNL	LogFewNL	basic
\LOGNP	LOGNP	full
\LOGSNP	LOGSNP	full
\LWPP	LWPP	full
\M	M	full
\MA	MA	basic
\MAC	MAC	basic
\MAE	MA-E	basic
\MAEXP	MA-EXP	basic
\mAL	mAL	basic
\MaxNP	MaxNP	basic
\MaxPB	MaxPB	basic
\MaxSNP	MaxSNP	basic

\mcoNL	comNL	basic
\MinPB	MinPB	basic
\MIP	MIP	basic
\MkP	$(\mathsf{M}_k)\mathsf{P}$	full
\mL	mL	basic
\mNC	mNC	basic
\mNL	mNL	basic
\mNP	mNP	basic
\ModkL	$\mathsf{Mod}_k\mathsf{L}$	basic
\ModkP	$\mathsf{Mod}_k\mathsf{P}$	basic
\ModP	ModP	basic
\ModZkL	$\mathsf{ModZ}_k\mathsf{L}$	full
\mP	mP	basic
\MP	MP	basic
\MPC	MPC	basic
\mTC	mTC	basic
\NAuxPDA	NAuxPDA	full
\NC	NC	basic
\NE	NE	basic
\NEE	NEE	basic
\NEEE	NEEE	basic
\NEEXP	NEEXP	basic
\NEXP	NEXP	basic
\NIPZK	NIPZK	full
\NIQPZK	NIQPZK	full
\NIQSZK	NIQSZK	full
\NISZK	NISZK	full
\NL	NL	basic
\NLIN	NLIN	basic
\NLOG	NLOG	full
\NP	NP	basic
\NPC	NPC	basic
\NPI	NPI	basic
\NPMV	NPMV	full
\NPMVsel	NPMV-sel	full
\NPO	NPO	full
\NPOPB	NPOPB	full
\NPSPACE	NPSPACE	basic
\NPSV	NPSV	full
\NPSVsel	NPSV-sel	full
\NQP	NQP	basic
\NSPACE	NSPACE	basic
\NT	NT	full
\NTIME	NTIME	basic
\OBDD	OBDD	full
\OCQ	OCQ	full

\Opt	Opt	basic
\OptP	OptP	basic
\p	p	basic
\P	P	basic
\PAC	PAC	basic
\PBP	PBP	full
\PCD	PCD	basic
\Pclose	P-close	full
\PCP	PCP	basic
\PermUP	PermUP	full
\PEXP	PEXP	basic
\PF	PF	full
\PFCHK	PFCHK	full
\PH	PH	basic
\PhP	PhP	full
\PINC	PINC	full
\PIO	PIO	full
\PKC	PKC	full
\PL	PL	basic
\PLF	PL	full
\PLL	PLL	full
\PLS	PLS	full
\POBDD	P-OBDD	full
\PODN	PODN	full
\polyL	polyL	full
\PostBQP	PostBQP	full
\PP	PP	basic
\PPA	PPA	full
\PPAD	PPAD	full
\PPADS	PPADS	full
\Ppoly	P/poly	basic
\PPP	PPP	full
\PPSPACE	PPSPACE	basic
\PQUERY	PQUERY	full
\PR	PR	full
\PrHSPACE	$\mathsf{Pr_H SPACE}$	full
\Promise	Promise	basic
\PromiseBPP	PromiseBPP	basic
\PromiseBQP	PromiseBQP	basic
\PromiseP	PromiseP	basic
\PromiseRP	PromiseRP	basic
\PrSPACE	PrSPACE	basic
\PSel	P-Sel	full
\PSK	PSK	full
\PSPACE	PSPACE	basic
\PT	PT	basic

\PTAPE	PTAPE	full
\PTAS	PTAS	basic
\PTWK	PT/WK	basic
\PZK	PZK	full
\QAC	QAC	basic
\QACC	QACC	basic
\QAM	QAM	basic
\QCFL	QCFL	basic
\QCMA	QCMA	basic
\QH	QH	basic
\QIP	QIP	basic
\QMA	QMA	basic
\QMAM	QMAM	basic
\QMIP	QMIP	basic
\QMIPle	$\mathsf{QMIP}_{\mathsf{le}}$	full
\QMIPne	$\mathsf{QMIP}_{\mathsf{ne}}$	full
\QNC	QNC	basic
\QP	QP	basic
\QPLIN	QPLIN	full
\Qpoly	Qpoly	full
\QPSPACE	QPSPACE	basic
\QSZK	QSZK	full
\R	R	basic
\RE	RE	basic
\REG	REG	basic
\RevSPACE	RevSPACE	full
\RHL	$\mathsf{R}_{\mathsf{H}}\mathsf{L}$	full
\RHSPACE	$\mathsf{R}_{\mathsf{H}}\mathsf{SPACE}$	full
\RL	RL	basic
\RNC	RNC	basic
\RNP	RNP	full
\RP	RP	basic
\RPP	RPP	full
\RSPACE	RSPACE	basic
\S	S	basic
\SAC	SAC	basic
\SAPTIME	SAPTIME	full
\SBP	SBP	full
\SC	SC	basic
\SE	SE	basic
\SEH	SEH	basic
\Sel	Sel	full
\SelfNP	SelfNP	full
\SF	SF	full
\SIZE	SIZE	basic
\SKC	SKC	basic

`\SL`	SL	basic
`\SLICEWISEPSPACE`	SLICEWISEPSPACE	full
`\SNP`	SNP	full
`\SOE`	SO-E	full
`\SP`	SP	full
`\SPACE`	SPACE	basic
`\spanP`	span-P	full
`\SPARSE`	SPARSE	basic
`\SPL`	SPL	basic
`\SPP`	SPP	basic
`\SUBEXP`	SUBEXP	basic
`\symP`	symP	full
`\SZK`	SZK	basic
`\TALLY`	TALLY	full
`\TC`	TC	basic
`\TFNP`	TFNP	full
`\ThC`	ThC	full
`\TreeBQP`	TreeBQP	full
`\TREEREGULAR`	TREE-REGULAR	full
`\UAP`	UAP	full
`\UCC`	UCC	full
`\UE`	UE	full
`\UL`	UL	full
`\UP`	UP	basic
`\US`	US	full
`\VNC`	VNC	full
`\VNP`	VNP	full
`\VP`	VP	full
`\VQP`	VQP	full
`\W`	W	basic
`\WAPP`	WAPP	full
`\WPP`	WPP	full
`\XORMIP`	$\mathsf{XOR\text{-}MIP}^*[2,1]$	full
`\XP`	XP	full
`\XPuniform`	$\mathsf{XP}_{\mathsf{uniform}}$	full
`\YACC`	YACC	full
`\ZPE`	ZPE	basic
`\ZPP`	ZPP	basic
`\ZPTIME`	ZPTIME	basic

4 Customization

The `complexity` package provides some 350 commands to typeset complexity classes. However, that should not mean that the commands here are the *only* ones you'll ever need. Expanding the list of commands to suit your needs is very easy. Please note, however, it is preferred that you not alter the base style file (`complexity.sty`). Instead, a file is provided for you to define your commands in (`mycomplexity.sty`).

4.1 Class Commands

To define a new complexity class, you can use the `\newclass` command which is similar (in fact is a macro for) the LaTeX command, `\newcommand`. The command takes two arguments: the command that you will use and how the class will be typeset. For example, say that we want to define the new complexity class, "VCCC" ("very complex complexity class"). We would use

```
\newclass{\VCCC}{VCCC}
```

Then, anytime we wanted to typeset our new class, we simply use `$\VCCC$`. Internally, `complexity` typesets everything using the command `\ComplexityFont` which is setup at the invocation of the package.

You also may have different preferences for typesetting the classes that `complexity` already defines. For instance, the class $\mathsf{promiseBPP}$ (typeset using the command `\promiseBPP`) is typeset with "promise" explicitly written. Preferring brevity over clarity, you may wish to typeset the same class as "pBPP". To do this, we use the `\renewclass` as follows.

```
\renewclass{\promiseBPP}{pBPP}
```

However, this only changes what the command does, not how we invoke it—we would still use `$\promiseBPP$`.

Consider a more complex example. Say we want to change how the class $\mathsf{Mod}_k\mathsf{L}$ (typeset using the command `\ModkL`) is typeset. By default, the subscript k is typeset in regular mathmode. We can change it so that it is typeset in the same font as the rest of the classes. We will have to specify this using `\renewcommand` as follows.

```
\renewcommand{\ModkL}{ %
  {
    \ComplexityFont{Mod}_{\ComplexityFont{k}}\ComplexityFont{L}
  }
}
```

Note the use of "extra" brackets. In your commands, more is always better (or at least safer); since we are using subscripts and superscripts, we want to ensure that if we use the `\ModkL` command itself in a subscript or superscript (say as an oracle) are typeset correctly.

4.2 Language Commands

You can define languages (to be typeset in the `langfont`) in a similar manner. Instead of using `\newclass`, however, you would use the command `\newlang`. You can also use `\lang` as a stand alone command in your document (i.e. `$\lang{Matching} \in \P$`) or you can define a command (using `\lang`) that can be reused throughout your document. Again, we give an example. Say we wanted to typeset the language "Graph Non-Isomorphism" using the abbreviation, "GNI". We could define something like the following.

```
\newlang{\GNI}{GNI}
```

In our document, we would would use something like `$\GNI \in \AM$`. We can also redefine any predefined language commands using the `\renewlang` command as before.

4.3 Function Commands

Again, the procedure for typesetting your own functions is the same as for classes. Here, however, you use the `\func` command. You can use it as a stand alone command (`$\func{lin}(n) \in \Theta(n)$`) or you can define a command that can be reused. Say we wanted to typeset a class of subexponential functions, say "subexp". We could define something like the following.

```
\newfunc{\subexp}{subexp}
```

In our document, we could then use `$\subexp(n) = 2^{o(n)}$`. We can redefine a function command using `\renewfunc`.

5 Extended Example

Here, we present an extended example using the package. Consider the following TEX code.

```
\documentclass{article}
\usepackage{complexity}
\begin{document}
It follows immediately from the definitions of $\P$ and $\NP$ that
$$\P \subseteq \NP$$
but the million dollar question is whether or not $\P
\stackrel{?}{=} \NP$.  As a generalization to these classes,
Stockmeyer (1976) defined a \emph{polynomial} hierarchy using
oracles.

\textbf{Definition}[Stockmeyer 1976] \\
```

```
Let $\Delta_0\P = \Sigma_0\P = \Pi_0\P = \P$. Then for $i > 0$, let
 \begin{itemize}
   \item $\Delta_i\P = \P$ with a $\Sigma_{i-1}\P$ oracle.
   \item $\Sigma_i\P = \NP$ with $\Sigma_{i-1}\P$ oracle.
   \item $\Pi_i\P = \coNP$ with $\Sigma_{i-1}\P$ oracle.
 \end{itemize}
Then $\PH$ is the union of these classes for all nonnegative
constant $i$.

It has been shown that $\PH \subseteq \PSPACE$. Moreover, Toda
(1989) showed the following
\textbf{Theorem}
$$\PH \subseteq \P^\PP$$
and since since $\P^\PP = \P^{\#\P}$ it follows that
$$\PH \subseteq \P^{\#\P}$$

\end{document}
```

Would produce something like the following:

It follows immediately from the definitions of P and NP that

$$\mathsf{P} \subseteq \mathsf{NP}$$

but the million dollar question is whether or not $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$. As a generalization to these classes, Stockmeyer (1976) defined a *polynomial* hierarchy using oracles.

Definition[Stockmeyer 1976]
Let $\Delta_0\mathsf{P} = \Sigma_0\mathsf{P} = \Pi_0\mathsf{P} = \mathsf{P}$. Then for $i > 0$, let

- $\Delta_i\mathsf{P} = \mathsf{P}$ with a $\Sigma_{i-1}\mathsf{P}$ oracle.
- $\Sigma_i\mathsf{P} = \mathsf{NP}$ with $\Sigma_{i-1}\mathsf{P}$ oracle.
- $\Pi_i\mathsf{P} = \mathsf{coNP}$ with $\Sigma_{i-1}\mathsf{P}$ oracle.

Then PH is the union of these classes for all nonnegative constant i.

It has been shown that $\mathsf{PH} \subseteq \mathsf{PSPACE}$. Moreover, Toda (1989) showed the following.

Theorem

$$\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$$

and since since $\mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\#\mathsf{P}}$ it follows that

$$\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$$

5.1 Acknowledgements

I'd like to thank Till Tantau for several useful suggestions and feature requests as well as some clever code segments for the `small` option. I'd also like to thank Enrico Gregorio for the suggested fix to disable redefinitions.