**Bol Processor BP2**

**Documentation**



Version 2.9.3 — October 1998

Bol Processor BP2 -- a 'QuickStart' 7

1. A few notes on BP2 7

1.1 Install BP2 9

1.2 Try it 9

1.3 Run a script 11

1.4 More notes 13

1.5 Performance controls 13

1.6 What else can be controlled? 15

1.7 Entering the arguments of performance controls from MIDI 16

1.8 Articulation 17

1.9 Parametric performance control 17

1.10 Ranges of performance control arguments 17

1.11 Changing note convention 18

1.12 Polyrhythmic sequences and polymetric expressions 19

1.12.1 The period notation 19

1.12.2 Polymetric expression 21

2. Sound-objects 22

2.1 Recording a sound-object 22

2.2 Putting together sound-objects 25

2.3 Other ways of creating sound-object prototypes 27

2.3.1 Importing a MIDI file 27

2.3.2 Capturing from text 27

2.3.3 Capturing from a tick cycle 28

2.4 Sound-objects, simple notes, homomorphisms 28

2.5 Cyclic sound-objects 29

2.6 Pre-roll and post-roll 30

2.6.1 Pre-roll 30

2.6.3 Post-roll 31

2.6.4 An important formula 31

3. Opening and compiling a grammar 32

3.1 What's in the examples folder? 32

3.2 Headers and related files 33

3.3 Organising projects 33

3.4 Network operation 35

3.5 A very simple grammar 36

3.6 A more elaborated project 37

3.7 Tokenised alphabet and grammar 38

3.8 Checking grammar consistency 40

3.9 The start string 40

4. Producing items 41

4.1 Listen to this one! 42

4.2 What's in a polymetric expression 45

4.3 The "Time base" dialog 46

4.4 Blowing memory on fractals... 49

4.5 Victor Hugo against Darius Milhaud 49

4.6 Graphic scale and time accuracy 52

4.7 Buffer size 53

4.8 The "Tuning" dialog 53

5. Non-stop improvisation 54

5.1 "Random" isn't anything... 54

5.2 A kind of cellular automaton playing a kind of Turkish music 55

5.3 The "INIT" instruction 57

5.4 Mozart's musical dice game 58

6. Interactive control 59

6.1 When does it react? 59

6.2 The MIDI filter 60

6.3 The "Interaction" window 61

6.4 Distributed improvisation 62

6.5 More interaction! 63

6.6 Synchronisation tags 64

6.7 Apple Events -- BP2 in a client-server environment 65

7. Glossaries 66

8. Scripts 67

8.1 Checking a script 68

8.2 Scripts, volumes and folders 68

9. Make it swing with smooth time... 68

9.1 Programming time-patterns 70

9.2 Loading time-patterns 72

9.3 Non-measured smooth time 73

10. For BP1 old-timers... 73

10.1 Tempo marking 73

10.2 Parsing items 74

10.3 Learning weights 75

10.4 Keyboard encoding 75

10.5 Documentation of Indian music 76

11. Programmed grammars 77

11.1 Using flags 77

11.2 New flag syntax 78

11.3 Classical problems solved with flags 79

12. Step computation 79

13. Manual time setting 82

14. Saving and loading MIDI code 82

14.1 MIDI files 82

14.2 Storing and sending MIDI data 83

15. More about windows... 83

15.1 The "Scrap" window 83

15.2 Adjusting window sizes and locations 83

15.3 BP2 editor 84

15.4 Remarks about remarks 85

16. What's in a "-se." file? 85

17. Csound on BP2 -- a primer 85

17.1 What is Csound, how does it relate to BP2? 85

17.2 Producing Csound scores with BP2 87

17.3 Try Csound with simple notes 88

17.4 Describing a Csound instrument 88

17.5 Checking the Csound output 92

17.6 Csound function tables describing waveforms 93

17.7 Sound-objects containing Csound scores 94

17.8 Creating a Csound score in a sound-object prototype 95

17.9 Performing musical items in the Csound environment 95

17.10 Performance controls and Csound scores 97

17.11 Combining argument values 98

17.12 Tabulated functions 98

17.13 Function tables in sound-object scores 99

17.14 A diagram of the cooperation between BP2 and Csound 100

**BP2 Reference Manual 101**

1. Task environment 101

2. Technical features 103

2.1 File description 103

2.2 File folders 103

3. Sound-object prototypes 103

3.1 Metrical properties of sound-objects 103

3.2 Topological properties of sound-objects 105

4. A short glance at the theoretical model 106

4.1 Variables and terminal symbols 106

4.2 Starting symbol 108

4.3 Patterns in BP grammars 108

4.4 BP2 grammars 108

4.5 Destroying structure 111

4.6 Weights 112

4.7 Metavariables 113

4.8 Tempo and gaps 113

4.9 Object concatenation 114

4.10 Polymetric expression 115

4.11 Period notation 120

4.12 Remote context 121

4.13 Metagrammars 122

4.14 Substitutions 123

5. The time setting of sound-objects 123

5.1 The time setting problem 123

5.2 How to proceed 126

6. Time quantization 127

7. "True" Bol Processor grammars and parsing 130

8. Procedures 131

8.1 Production procedures 131

8.2 Trace procedures 132

8.3 Renumbering and remapping conditional jumps 132

9. Csound argument mapping 134

**Bibliography 135**

**Appendix 137**

1. Solving problems 137

1.1 Using the printer serial output 137

1.2 Inter-application communication 137

1.3 Memory 137

2. Poem "Les Djinns" 138

3. Disclaimer 140

4. Acknowledgements 140

**Index 141**

**Bol Processor BP2**

 **'QuickStart' and reference manual**

This document does NOT contain information about important recent features such as \_tempo, \_retro, \_keymap, \_transpose, etc., all of which are documented on-line in BP2 and introduced in <http://perso.wanadoo.fr/bel/music/whatsnew.htm>.

This doc has been converted to HTML (using **rtftohtmlppc**) and may be accessed on BP2 site: <http://perso.wanadoo.fr/bel/music/index.html>

In the early 1980s, ethnomusicologist Jim Kippen and I developed a computer system called the *Bol Processor* (i.e. "BP") to help us examine improvisatory methods used by North Indian tabla drummers (Kippen 1987, 1988, Kippen & Bel 1989b, 1989c). Designed for the portable Apple IIc, the rule-based BP1 was able to analyse musical input (in the form of tabla *bol*s, or onomatopoeic syllables) as well as generate new improvisations that musicians could assess. Details of the philosophy behind BP1 and its *modus operandi* can be found in our paper: "Modelling music with grammars: formal language representation in the Bol Processor" (Kippen & Bel 1992).

The Bol Processor attracted interest from scholars and musicians alike. It was felt that the formal model embedded in it could be expanded to encompass more general musical structures, and in this form could be of some benefit as a tool for music composition. We therefore decided to implement an alternative version of the Bol Processor on the Apple® Macintosh. An enhanced version of the algorithmic part of the Bol Processor, namely "BP2", was written in the C language. A sound-object editor allowed it to interact with MIDI devices.

BP2 deals with incomplete representations of polyphonic/polyrhythmic musical structures. It uses a very efficient algorithm for the synchronisation of sound-objects.

With version 2.5 important features have been implemented, notably an accurate control of all MIDI parameters and automation of tasks with scripts. Version 2.6 introduced client-server technology. The present version is interfaced with Csound, deals with MIDI files, and works under Opcode OMS.

BP2 won the Bourges 1997 international award (ex aequo with Piché & Burton's "Cecilia") in the category of computer-aided composition and realization software. It is currently jointly developed by Bernard Bel <bel@kagi.com> and Srikumar Karaikudi Subramanian <srikumar@krdl.org.sg>.

BP2 may be downloaded from Info-Mac mirror sites (list available under <ftp://ftp.amug.org/pub/mirrors/info-mac/help/>) or from its home web site: <http://perso.wanadoo.fr/bel/music/index.html>.

**Bol Processor BP2 -- a 'QuickStart'**

This is a short introduction to **Bol Processor BP2**.

Since BP2 uses the standard Apple Macintosh® interface, any person familiar with common software (e.g. word processors) can operate it easily.

BP2 is thoroughly documented on-line: type cmd-? (question mark with the command key down) and click any button, menu item, static text or word in a text. You may also open the text file "BP2 help" with any word processor.

The present introduction is meant to explain how BP2 works and help you to start designing your own (presumably musical) sounds.Throughout the introduction we will use files from the example folders so that you do not need to create and debug grammars, alphabets, sound-object files, etc., before seeing and hearing BP2 in operation. The reference manual (below in this document) is more informative about the way BP2 handles musical syntax.

**1. A few notes on BP2**

BP2 is able to interact simultaneously with three environments: **MIDI**, **QuickTime Music** and **Csound**. The latter is presented in §17.

To produce MIDI or QuickTime music with BP2 you need either of the following setups:

• a MIDI interface connecting the 'modem' or 'printer' serial port of the Macintosh to a **MIDI synthesizer, sampler or expander**. This interface is a cheap item available in music or computer shops. A few expanders have built-in MIDI interfaces and can be hooked directly to the serial port. If your Mac has a unique serial port connector (e.g; PowerMac 6500 or a cheap PowerBook) you may be forced to install OMS (see below) to enable communication with MIDI. On some PowerBooks (such as my 5300cs) this serial port becomes a valid 'modem' port when either EtherTalk or IRTalk is active, but old PowerBooks such as the PB 150 will remain dumb unless OMS is properly installed.

 *This setup will allow you to produce musical pieces with a high sound quality depending on external hardware.*

• .i.Opcode OMS; (.i.Open Music System;) installed and an OMS-compatible **software synthesizer** (such as SoftSynth) running on the same machine as BP2. The MIDI output of BP2 will be routed to the synthesizer via an .i.**Inter-Application Communication (IAC); bus** handled by OMS. (See OMS documentation) Beware that your Mac is powerful enough, and has enough RAM, to run BP2 simultaneously with the synthesizer.

 *This solution also produces high sound quality. It is recommended to connect the audio output to an external amplifier.*

• .i.**QuickTime Music**; and Opcode OMS installed. QuickTime Music is a standard extension of MacOS. Run the "Extension Manager" control panel to make sure that QuickTime extensions are enabled at startup. In system versions below MacOS 8.0, you will also need an extension named ".i.QuickTime® Musical Instruments;" in the Extensions folder.

 Opcode OMS is free of charge and supplied with many music software packages. You may download it from <http://www.opcode.com>. You must use a version equal or greater to 2.0.

 *With this setup you will be able to play BP2 music on the Mac's loudspeaker, using the soft synthesizer imbedded in QuickTime.*

 If you connect the Mac to an amplifier and speakers you will notice that the sound quality is far from that of a professional synthesizer. However, this quality is bound to improve with each upgrade of QuickTime Music.

The preceding setups may of course be combined. Once OMS has been installed you will be able to use its setup to direct BP2's output to the device of your choice in a MIDI studio, or to the internal loudspeaker via QuickTime Music. Again, installing OMS is highly recommended if you plan to spend some time with this and other MIDI music software.

Once you have understood what BP2 is about you will also be able to .i.**export music**; in two formats:

• **MIDI files** that may imported and manipulated by any MIDI software (including BP2 itself). Check ".i.MIDI files;" and uncheck "MIDI" in the "Devices" menu;

• **Csound scores** that may be used to produce sound files in the Csound environment. BP2 has advanced features enabling you to handle sophisticated Csound instruments in the most flexible and user-friendly way. If you are already conversant with Csound and your Mac has a 16-bit sound editor you may check ".i.Csound scores;" in the "Devices" menu and proceed immediately to §17.

MIDI instruments (keyboard, guitar, drum, etc.) may optionally be used to control BP2.

Conventional sounds like cello, piano, flute, drums, etc., will be good enough for experimenting with BP2. Some of the supplied examples will indeed sound bizarre unless they are performed on the MIDI device they were designed for: a Roland® D-50 or D-550 synthesiser with patch data ROM PN-D50-02 and Musitronics Multitimbral Expansion card (phew!).

A few sound examples are attached to a paper available on line (see <http://www.fortunecity.com/victorian/dada/181/jim96/jim96.htm>). In the near future, more musical examples using common MIDI hardware, QuickTime instruments and Csound will be made available for demos.

BP2 is designed to run on a 14-inch monitor, i.e. minimum 640 x 480 pixels, with at least 16 gray or color levels.

**1.1 Install BP2**

BP2 is a stand-alone application running on Macintosh® configurations using a 68020, 68030, 68040 processor, PowerMac or G3. It does not require an arithmetic coprocessor. It is currently being developed and checked under MacOS 8.1 and will not run with system versions below 7.1. The PowerMac optimized version is called ".i.BP2.9.3 (ppc);" and the 680x0 one ".i.BP2.9.3 (68k);". We check every version under MacOS 8.1 on a "slow" Mac (PowerBook 5300cs) and a "fast" one (G3 / 333MHz).

To run "BP2.9.3 (ppc)" on a PowerMac, first drag the file "OMSLibPPC.slb" to the System folder, unless it has already been put there when installing another MIDI program. This file is supplied with BP2 in a folder named "Drag content to System". It is the shared library of OMS which BP2 requires even when it is not using OMS. The file will reside in the "Extensions" folder.

BP2 requires no other system extension or control panel. Its default settings file ".i.-se.startup;" and its on-line help file ".i.BP2 help;" must reside in the same folder as the program, not in the System folder.

If you still own an old version of BP2 you should trash it, install the new one and restart the computer holding down the 'command' (cmd) and 'option' keys down to rebuild the desktop. Keeping several versions of BP2 may lead to unpredictable situations. Do not hesitate to trash obsolete versions because files get automatically upgraded to the new version.

**1.2 Try it**

If you wish to run BP2 under OMS, first launch the .i.**OMS Setup**; utility and make sure that the required output runs fine. When you setup OMS the external MIDI device(s) must be plugged to the Mac and switched on, so that OMS will sense them automatically and create the appropriate studio setup document accordingly. Needless to say, the MIDI interface should also be powered on.

If the IAC Driver used for Inter-Application Communication is not displayed on the document, you should include it. For this, quit OMS Setup and run the customized installation of OMS.

The studio document should also display the .i.**QuickTime Music**; icon. If the icon is crossed, double-click it and check the box of QuickTime Music. Clicking the icon also allows you to set the pitchbend range to 2 semitones, the value used in examples of this document.

If a MIDI device (keyboard, synth...) is connected to the MIDI interface attached to a serial port, the icon of the port ('modem' or 'printer') should be dispayed along with a line connecting it to the icon representing the device.

Type cmd-T (command-T) to activate "Test studio" and click the icons of QuickTime Music and other output devices. You should hear chords sent by OMS to these devices. Playing a MIDI keyboard should similarly produce a visual effect on the studio document.

If there is no communication, select ".i.**OMS MIDI setup**;" in the "Edit" menu to select the proper serial port.

In OMS MIDI setup you shoud also check "Run MIDI in background" as BP2 may be used as a server.

When done with all this, save the studio document and quit OMS Setup.

Users of MIDIshare should make sure that messages are routed via OMS. BP2 does not recognise MIDIshare and will communicate with it only via OMS.

If you try BP2 to use OMS while OMS is not active on your system, you will get an explicit error message and BP2 will switch to its own built-in driver, which has limited features.

At startup, BP2 reads information about the default OMS input and output stored in the ".i.-se.startup;" file. This file is supplied by default with the setting of QuickTime Music as the output, and no input.

Double-click BP2.9.3 (68k) or BP2.9.3 (ppc), depending on the type of computer you are using. In the welcome alert, click "Data", then click the button "Check MIDI" on the Control panel, and take the option "Check output." A dialog with 16 buttons resembling a calculator is displayed. Click button '1' to play random music on channel 1.

You may not hear anything at all. No panic! Try the following procedures:

• If BP2 displayed a message telling that it failed to sign in with OMS, you must install and activate OMS, unless you wish to use the in-built MIDI driver. If BP2 did not complain about anything, it means that the MIDI interface, the synthesizer and/or the amplifier are wrongly connected, not powered, communicating via the wrong MIDI channel, or set to volume zero... Some MIDI interfaces such as *Opcode MIDI Translator II* have input/output lights that keep flashing when data is transmitted. This helps tremendously.

 By the way, don't forget to check the volume pedal on the synthesizer! I have spent hours finding it...

• OMS may be installed and yet inactive because the "OMS" extension or the ".i.OMS Preferred Device;" (plus perhaps more extensions and control panels in future versions of OMS) are inactive. Check this in the "Extension Manager" control panel and restart your Mac.

• If OMS is active you won't get any warning at startup and OMS items will become active (displayed in black) in the "Devices" menu. Select ".i.OMS input-output;" and pull down the output menu to activate ".i.QuickTime Music;". If you have gone this far it generally works.

• If OMS is active but you cannot find ".i.QuickTime Music;" in the output list, select ".i.OMS Studio setup;". If "QuickTime Music" appears crossed out on the studio document, double-click it and switch in on.

• If you expected BP2 to play sounds on an external MIDI device, and OMS is active, select ".i.OMS input-output;" and check the name of your MIDI device. If it is not in the list, run ".i.OMS Studio setup;".

If you still don't hear it, run ".i.OMS MIDI setup;" to select 'modem' or 'printer'. This may require entirely switching off the MIDI interface, the synth and the Mac, including disconnecting the power supply from a PowerBook... because the serial port, the interface or the input of the synthesizer might be stuck.

• If OMS is not active, you are using the in-built MIDI driver. All you need is to check either 'modem' or 'printer' in the "Devices" menu, assuming, on certain Macs, that you run IRTalk or EtherTalk instead of LocalTalk. (See the "AppleTalk" or "Network" control panel.) Remember, this will not work on some old PowerBooks such as the PB 150, which cannot communicate with MIDI unless OMS is installed.

• If OMS is active and BP2 runs in background, for example you launch it from a FileMaker Pro database such as ".i.TryAppleEvents.fm;", you will not hear any sound unless ".i.Run MIDI in background;" is checked on the ".i.OMS MIDI setup;" panel that is accessible from the "Devices" menu. Since it is an application-independent setup this option will remain set until you decide to change it.

Once you have performed changes in the ".i.OMS input-output;" menus, these may not work immediately. In any case you should store these changes to the default settings file. Select ".i.**Modify '-se.startup'**;" in the "File" menu. BP2 remembers where it found the startup settings file and will replace it with the updated one. If communication is not working, quit and restart BP2.

If the MIDI output is still dumb... There are bad days like this, notably when a group of enthusiasts or a person you invited for a candle dinner is sitting around waiting for a demo! Check cables, power supplies, amplifiers, speakers and the synthesiser's MIDI set-up until you hear sounds. (Maybe your device is not receiving on the proper MIDI channel? Check the channel number, set .i.OMNI ON;, or else, just call your little brother who knows all about such things...) *Et voilà*!

Don't invite your 'windoze' neighbour over a show until you can manage to make sounds. Otherwise s/he'll feel like saying something silly, for instance "my SoundBlaster card works better!"...

**1.3 Run a script**

(You can send your brother back to his video game now. Here comes some substantial stuff.)

Quit BP2 and double-click the ".i.+sc.TryMe;" file. BP2 will run this demo script which plays a short burst of music on the MIDI device (channel 1) or the loudspeaker (via QuickTime music) and prompts the user to hit a key of the MIDI keyboard or click the mouse.

You may skip the rest of this section if you are not enthusiastic about AppleScript and any programming stuff. Instead you might try working with simple notes (see §1.4), performance controls (§1.5), sound-objects (§2) or grammars (§3-4).

Type cmd-– or select "Script" in the "Windows" menu. The following script is displayed:

// This is a script for Bol Processor BP2

Show messages OFF -- avoid useless display

Note convention = English

Show graphics ON

Play {2,G4,E4,C4,- C5 - C5 {C5\_-,A#4\_ \_}}

Show messages ON

Activate window Scrap

Type Welcome to Bol Processor BP2...

Type <return>

Type <return>

Type Hope you enjoyed the music!

Type <return>

Type <return>

Type If you didn't hear anything, select 'QuickTime Music' for the output in 'OMS input-output', then 'Modify -se.startup' in the 'File' menu. If Opcode OMS is not installed, or you don't know what OMS stands for, read the documentation...

Type <return>

Type <return>

Type The documentation is in the 'bol-processor-xxx-doc' folder, and examples in 'bol-processor-xxx-data'. ('xxx' is the version number) You must download these folders to use BP2.

Type <return>

Type <return>

Type <return>

Type BP2 is now waiting for you to play 'C4' on the MIDI keyboard, or click the mouse...

Wait for C4 channel 1

Script instructions can be created in three ways:

• typing them directly into the .i."Script" window;;

• "pulling down" an instruction from the .i."Script" menu; and filling arguments between « quotes » with acceptable values. (Use sticky menus in MacOS 8.0 to facilitate the selection)

• recording actions in script form while they are being performed in BP2, a feature similar to AppleScript. To set the script manager to recording mode, check the ".i.REC;" button on top of the .i."Script" dialog;.

Instructions obtained by pulling down the .i."Script" menu; are hilited below. The remaining instructions were recorded in the "REC" mode.

// This is a script for Bol Processor BP2

**Show messages OFF** -- avoid useless display

**Note convention = English**

**Show graphics ON**

**Play** {2,G4,E4,C4,- C5 - C5 {C5\_-,A#4\_ \_}}

**Show messages ON**

Activate window Scrap

Type Welcome to Bol Processor BP2...

Type <return>

Type <return>

Type Hope you enjoyed the music!

Type <return>

Type <return>

Type If you didn't hear anything, select 'QuickTime Music' for the output in 'OMS input-output', then 'Modify -se.startup' in the 'File' menu. If Opcode OMS is not installed, or you don't know what OMS stands for, read the documentation...

Type <return>

Type <return>

Type The documentation is in the 'bol-processor-xxx-doc' folder, and examples in 'bol-processor-xxx-data'. ('xxx' is the version number) You must download these folders to use BP2.

Type <return>

Type <return>

Type <return>

Type BP2 is now waiting for you to play 'C4' on the MIDI keyboard, or click the mouse...

**Wait for** C4 channel 1

The first line is a comment preceded by the double slash // familiar to C programmers. The first instruction (second line) contains a remark preceded by "--", as in AppleScript or HyperTalk. The next instruction tells BP2 that .i.**simple note;s** shall be notated in the English convention. (See other conventions §1.11)

The instruction "Play..." tells BP2 to play a musical item on the selected output. Here the item is polyphonic and notated as a .i.**polymetric expression**;. You will learn more about polymetric expressions in §4.2 of this introduction, or §4.10 of the reference manual. In brief, the digit "2" indicates that the expression's length is 2 beats, while other **fields** separated with commas denote the four "voices". We have a chord {G4, E4, C4} sustained throughout the entire item, as well as a more complex melodic phrase "- C5 - C5 {C5\_-, A#4\_ \_}" which contains another polymetric expression. Symbols '-' and '\_' indicate .i.**silence**;**s** and .i.**note prolongation**;**s** respectively. Thus, "C5\_" could be worth a half-note and "C5" a quarter-note.

Other instructions in this script are self-explanatory.

**1.4 More notes**

The ".i.+sc.TryMe;" script demonstrated BP2's ability to deal with ".i.**simple note;s**" in the same way most MIDI programs do. BP2 was initially designed to handle ".i.**sound-object**;**s**" containing arbitrary sequences of elementary events (MIDI messages or Csound score lines), and it still handles them. You will learn about sound-objects very soon. Meanwhile, here is an overview of things you can do with simple notes.

Type cmd-d or select "Data" in the "Windows" menu. Then type cmd-j or select ".i.Type from MIDI;" in the "Action" menu. The cursor changes to a small keyboard: . From now on, any note played on a connected MIDI input device will print the note name at the current cursor position. It uses the convention selected in the "Misc" menu: English, French-Italian, Indian or Key numbers.

Notes may be transposed if ".i.Transpose input notes;" has been checked on the Control panel and a non-zero number of semitones has been entered near the check box.

In the "" mode, pushing the "hold" pedal inserts a one-beat prolongation '\_'. There is a smarter way of entering durations: see the end of §4.3.

**1.5 Performance controls**

MIDI folks are primarily interested in controlling all sound parameters in a flexible and acurate way. A Japanese company is already offering direct control of MIDI by brainwaves! Perhaps the reverse -- controlling the brain from MIDI -- would make more sense, but this is what music is about after all.

Access to fine control may be compared with skills at playing an instrument. In BP2 the term .i.**performance control**; is used to name an access to such parameters.

The following is a demo of BP2's ability to control MIDI parameters. Detailed comments and sound examples may be found in (Bel 1996a) available from <http://www.fortunecity.com/victorian/dada/181/jim96/jim96.htm>.

Enter a sequence of notes using the MIDI keyboard, then select it with the mouse and type cmd-p, or select "Play selection" in the "Action" menu. You will hear the sequence just entered. Durations and velocity information are ignored. They can be specified afterwards. Notes are played with the current metronome setting (mm = 60 by default). Type cmd-m to change the speed.

Readers who never heard, and don't want to hear, about pitch benders and other .i.MIDI torture; instruments might jump to §1.11.

Let us now edit the sequence. Suppose that you entered "C5 D5 F5 G5 C6 A#5 A5 G5". Type cmd-j once again to return to text editing mode (the blinking cursor). Then try the following .i.performance control;s:

(These examples belong to file ".i.-da.SomeNotes;" that may be opened from the "Data" window.)

• Silences and prolongations: C5 - D5 F5 G5\_ C6 A#5 A5 G5

• Polyphony (see §1.12): {C5 D5 F5 G5 C6, A#5 A5 G5}

• Imbedded script instruction: \_script(Beep) C5 D5 F5 G5 C6 \_script(Wait for space) A#5 A5 G5

• Performance control (velocity): \_vel(60) C5 D5 F5 G5 \_vel(127) C6 A#5 A5 G5

• Performance control (crescendo): \_volumecont \_volume(20) C5 D5 F5 G5 \_volume(127) C6 A#5 A5 G5 \_volume(80)

• Performance control (pitchbend): \_pitchcont \_pitchrange(200) \_pitchbend(0) C5\_\_\_ \_pitchbend(150)\_\_\_ \_pitchbend(-180)

• Program change: \_script(MIDI program 1) C5 D5 F5 G5 \_script(MIDI program 5) C6 A#5 A5 G5

The last example deserves a comment: ".i.\_script(MIDI program x);" selects MIDI program *x* on the MIDI device, with the effect of changing the timbre (the "patch"). This generally requires a set-up time on the synthesiser, roughly 0.5 second on my D-50 and much more with QuickTime Music. It is therefore not advisable to insert this instruction in a fast movement. It is generally better to use a multitimbral patch able to assign a different instrument to each MIDI channel. .i.QuickTime Music; does it fine; see ".i.MIDI orchestra;" in the "Devices" menu and follow on-line help.)

Performance control "\_script(MIDI program...)" may be recorded by BP2 in the same way notes are entered: type cmd-j and change the patch manually on the MIDI device. This will paste the instruction to the current edit window.

Instead of typing .i.performance control;s you may pick them from a predefined list: select ".i.Pick performance control;" in the "Edit" menu.

If you want to know everything about .i.**performance control;s** in BP2, you may activate the "Data" window and load ".i.-da.checkControls;". (Type cmd-o or select "Open" in the "File" menu.) There is a line ".i.-gl.GeneralMIDI;" on top of the window, meaning that BP2 is going to use a .i.**glossary**; comprehensible to synthesisers based on General MIDI.

To hear all examples, type cmd-a or select "Select all" in the "Edit" menu, then type cmd-p or select "Play selection" in the "Action" menu. BP2 skips remarks and plays all items in the window.

**1.6 What else can be controlled?**

MIDI specifications make it possible for BP2 to handle **continuous** changes of the following parameters:

• **Pitch**, **modulation**, **channel pressure** (polyphonic aftertouch), **volume** (default controller 7) on all synthesisers;

• **Panoramic** (default controller 10) on synthesisers complying with .i.**General MIDI**;;

• **Up to 250 parameters with arbitrary names** that are only recognised by Csound instruments. See ".i.\_value;(param,x)", ".i.\_step;(param)", ".i.\_cont;(param)" and ".i.\_fixed;(param)" in §17.5.

The sampling rate (default 50 messages per second) can be adjusted to compromise between accuracy and MIDI bandwidth. Keep in mind that BP2 creates messages in real time, sometimes resulting in hundreds of messages being sent to MIDI every second. Some of them might get lost notably if other MIDI devices are active on the same network. When this happens it is necessary to reduce sampling rates to the minimum values required for smooth movements. Ten messages per second would be enough in many cases.

.i.**Velocity**; is not controlled continuously: a unique velocity value is assigned to every note or sound-object. (In the case of sound-objects it is possible to instruct some objects to ignore this velocity assignment.) Velocity values may be interpolated between two pre-set values along the sequence. This is done using instruction ".i.\_velcont;" (or equivalently ".i.\_velstep;") which may be cancelled by ".i.\_velfixed;".

.i.**Transposition**; is supported by the ".i.\_transpose;(x)" command in which *x* is the number of semitones up (down if negative). Note that transpositions may be cumulated, using curled brackets (i.e. .i.polymetric expression;s). For instance,

 \_transpose(-3) { A4 \_transpose(+7) C3 D3}

is interpreted as:

 A4 \_transpose(+4) C3 D3 [ ... since -3 + 7 = 4 ]

Controls are limited by MIDI specifications. For instance, it is not possible to assign different pitchbend corrections to several notes on the same MIDI channel. Fanatics of microtonal intonation must either content themselves with monodic music or distribute notes on separate channels.

Each MIDI implementation has its own limitations: for instance, Roland D-50 does not distinguish volume controls on separate MIDI channels, even though BP2 produces the correct messages.

Results also depend on the .i.**OMNI ON/OFF**; and .i.**POLY/MONO**; settings, briefly discussed in ".i.-da.checkControls;". See MIDI literature or the synthesiser's instruction manual for a detailed explanation. BP2 makes it possible to change these settings with script commands. These must be executed on the basic channel of the MIDI device. See the instruction manual of the device.

MIDI implementation are by-passed by some software environments. For instance, Opcode OMS makes it possible to use more than 16 MIDI channels; it simply maps channels above 16 to other devices. Future versions of BP2 will take advantage of such features.

Csound is another way of going far beyond MIDI. (See §17)

**.i.1.7 Entering the arguments of performance controls from MIDI**

In the same way notes can be "typed" from the MIDI keyboard, the arguments of **.i.performance control;s** may be captured from a MIDI device. Suppose that you entered "C5 D5 F5 G5 C6 A#5 A5 G5" in the "Data" window. Select ".i.Pick performance control;" in the "Edit" menu and insert the following controls (or load file ".i.-da.checkCapture;"):

\_chan(•) \_pitchrange(•) \_volumecontrol(•) \_volume(•) \_vel(•) C5 D5 F5 \_vel(•) G5 C6 \_mod(•) A#5 \_pitchbend(•) G5 F5

Arguments marked by a period need to be replaced with numeric values. Type "1" for the argument of ".i.\_chan;" and "200" for that of ".i.\_pitchrange;", assuming that the pitch bender of the MIDI device is set to vary ± 200 cents, or another value if it is set differently. (Each semitone is 100 cents.)

Type cmd-j. The cursor takes its "keyboard" shape . Click the word "\_pitchbend". The argument of ".i.\_pitchbend;()" gets hilited. Now move the pitch bender. BP2 automatically converts in-coming pitchbend messages to cent values (within the specified range) writing the result as an argument of \_pitchbend. Of course it is a good idea to play A5 or G5 while you are moving the pitch bender, so that you can hear the effect.

This process works if the pitch bender is sending messages to the specified channel. Otherwise, click "\_chan", move the bender, and let BP2 record the channel number.

Hmmm. Great, but if I release the bender the final value is always 0, isn't it? The solution is to hit the space bar or click the mouse anywhere (outside the word "\_pitchbend"...) as soon as the desired position has been reached.

Do the same with modulation: click ".i.\_mod;" and move the modulation bender while playing notes. You can adjust the argument for ".i.\_vel;(•)" similarly: BP2 records velocities of in-coming NoteOn's.

For volume, first click "\_volumecontrol" and move the volume pedal to record its number (most likely 7). Then click "\_volume" and record the pedal position.

.i.Performance control;s whose numeric arguments may be interactively entered are:

\_chan(), \_vel(), \_mod(), \_press(), \_volumecontrol(), \_pancontrol(), \_volume(), \_pitchbend(), \_pan().

Values recorded in the last three ones depend on preceding ".i.\_volumecontrol;()", ".i.\_pitchrange;()" and ".i.\_pancontrol;()" instructions respectively. Instructions ".i.\_volumecontrol;(x)" and ".i.\_pancontrol;(x)" instruct BP2 that controller x is the one modifying volume or panoramic respectively. Default arguments are 7 for volume and 10 for panoramic.

**1.8 Articulation**

Data file ".i.-da.checkArticulation;" demonstrates BP2's ability to deal with .i.*legato;* and .i.*staccato*;. The whole set of examples may be played with a "Cello" sound.

Articulation values may be interpolated between two pre-set values, given that ".i.\_staccato;(x)" is equivalent to ".i.\_legato;(-x)" and conversely.

**1.9 Parametric performance control**

For all .i.performance control;s taking an argument **in range 0..127** it is possible to replace the argument with a .i.**control parameter**; K1, K2,... K32767. For instance, ".i.\_vel;(K3 = 60)" will assign velocity 60 and set K3 to the same value. If later, for instance, \_press(K3) is found, then channel pressure will be set to the current value of K3. These control parameters are therefore global numeric variables.

Parameters K1, K2... may also be used as .i.**weights in grammar;s** (see §5.4 or reference manual §4.6) and may even be modified by MIDI controllers or NoteOn velocities, depending on infomation given in the "Interaction" window, for instance:

IN Param K3 = controller #18 channel 1

(See §6.5 for more detail)

**1.10 Ranges of performance control arguments**

The following ranges are imposed by MIDI specifications.

• \_chan(1..16)

• \_legato(0..127)

• \_mod(0..16383) -- some devices may ignore the least significant byte

• \_modrate(0..16383) -- samples per second

• \_pan(0..127)

• \_pancontrol(0..127) -- index of MIDI controller

• \_panrate(0..16383) -- samples per second

• \_pitchbend(0..16383) if not preceded by \_pitchrange(r),
otherwise (-r..+r) with *r* in cents

• \_pitchrange(0..16383) -- this value is generally lower and depends on the synthesiser: "\_pitchrange(r)" instructs BP2 that the range of the pitch bender on the current MIDI channel is (± r) cents. It sets the limit of the argument of \_pitchrange() to the same value.

"\_pitchrange(0)" indicates that subsequent "\_pitchbend(x)" instructions take their argument *x* as the actual MIDI pitchbend position (range 0..16383) instead of a value expressed in cents.

• \_pitchrate(0..16383) -- samples per second

• \_press(0..127)

• \_pressrate(0..16383) -- samples per second

• \_staccato(0..127) -- in fact the value is clipped to the 0..100 range.

• \_transpose(-128..+127) -- semitones

• \_vel(0..127)

• \_volume(0..127)

• \_volumerate(0..16383) -- samples per second

• \_volumecontrol(0..127) -- index of MIDI controller

**1.11 Changing .i.note convention**

There are four options for note conventions in the "Misc" menu: "English" (default), "French", "Indian" and "Key numbers". The convention is saved in the "-se.<filename>" settings, the name of which may appear on top of a grammar or data file. The default convention is saved in ".i.-se.startup;".

Changing the convention brings to front a keyboard dialog, for instance the one shown Fig.1 (French-Italian convention).



**Fig.1 Changing the note convention to "French-Italian"**

The selected names will be the ones used for displaying items. Alternate names (e.g. "reb" instead of "do#") are also recognised when parsing grammars, data or scripts. The selection of names is saved in the "-se.<filename> " settings file.

Octaves are numbered as per the convention:

middle C = C4 = do3 = sa4 = MIDI key number 60

If A4 = la3 = dha4 = 440Hz, then C0 (MIDI key 12) is approximately 16Hz whereas C1 or do1 (MIDI key 24) comes to about 32Hz (*le do grave du violoncelle*). Since these frequencies are low, MIDI keys 0 to 31 are rarely used for tonal data, but they might be reserved to other types of messages. The octave starting with MIDI key 0 is notated:

C00 Db00 D00...

do000 reb000 re000...

sa00 rek00 re00

whereas the one starting with MIDI key 12 is notated:

C0 Db0 D0...

do00 reb00 re00...

sa0 rek0 re0

This convention was not the same in previous versions of BP2. See §4.8 for customizing it.

**1.12 Polyrhythmic sequences and polymetric expressions**

Do not skip this section! It introduces one of the most useful features of BP2, and examples are easy to perform. (Lazy readers will find them in file ".i.-da.PeriodNotation;".)

Display the "Data" window and type cmd-j or select ".i.Type from MIDI;" in the "Action" menu (see §1.4). Now, play a few notes on the MIDI keyboard. These are immediately transcribed to the "Data" window, for instance:

C5 D5 G5 F5 G5 C5 D5 D#5 D5 C5 A#4 C5

Now select this sentence and type cmd-p. A dull performance, isn't it? Keep in mind that BP2 allows all kinds of manipulations of note durations (see §1.8) and time patterns (see §9). For the moment we will use a plain metronome to demonstrate rhythmic possibilities at the level of the *score*. We call this dealing with .i.**symbolic durations**;.

1.12.1 The .i.period notation

Insert periods (indicating beats), for instance:

C5.D5.G5.F5 G5.C5 D5.D#5.D5.C5 A#4.C5

Play this example. Symbolic durations have been resized automatically so that the sum of durations remains constant over each 'beat'. In western terminology, the item would be said to contain crotchets and quavers. In Indian terminology, the same would be described as tempo changes from *hargun* (speed 1) to *dogun* (speed 2).

This .i.**period notation**; is a recent feature of BP2. Previously, the simple (although less flexible) way of notating speed changes was:

/1 C5 D5 G5/2 F5 G5 C5 D5/1 D#5 D5/2 C5 A#4/1 C5

A slash followed by an integer number specifies the .i.**tempo**;, more precisely the .i.**density of sound-objects**; (in units per beat). We call it an .i.**explicit tempo marker**;. Tempo markers are still used in the present version, and may be combined with the period notation. Read on...

You may try all kinds of arrangements of periods on a sequence. You will find that changes of tempo may introduce fractional ratios. Try for instance:

C5 D5.G5 F5.G5 C5 D5.D#5 D5 C5.A#4 C5

Here, the duration of each note in "G5 C5 D5" and "D#5 D5 C5" is 2/3d that of the first note "C5". Is it possible to represent this example with explicit tempo markers? The answer is given by BP2: select the item and click "Expand selection" on the Control panel (accessed with cmd-=). The explicit notation is the following:

/3 C5\_ \_.D5\_ \_.G5\_ \_.F5\_ \_.G5\_ C5.\_ D5\_.D#5\_ D5.\_ C5\_.A#4\_ \_.C5\_ \_

in which '\_' indicates a prolongation of the preceding note. The initial part "/3 C5\_ \_" means that the tempo is "speed 3" (*tigun* in Indian notation), but since "C5" is followed by two prolongation symbols its resulting duration (3/3 = 1) is the same as in the preceding examples.

There is an important rule regarding tempo: in a sequence, the tempo of sound-objects or notes in the first beat sets the initial tempo, whatever the beat division. The initial tempo may be the default one "/1" unless otherwise specified. Examples will make it clear. First we introduce an explicit initial tempo:

/2 C5 D5.G5 F5.G5 C5 D5.D#5 D5 C5.A#4 C5

Now, something more tricky: although the third beat "G5 C5 D5" contains three notes we want it to be performed at "speed 2". This will force BP2 to resize other beats accordingly. The notation is:

C5 D5.G5 F5./2 G5 C5 D5.D#5 D5 C5.A#4 C5

What is the duration of the initial "C5" in this case? It is 3/2 that of each note in the third (and fourth) beats. Since the latter are forced to speed 2, the resulting duration for "C5" is 3/2 x 1/2 = 3/4. This is shown by the expanded notation of the same example:

/4 C5\_ \_ D5.\_ \_ G5\_.\_ F5\_ \_.G5\_ C5\_.D5\_ D#5\_.D5\_ C5\_.A#4\_ \_ C5.\_ \_--

Note that for the sake of consistency BP2 completed the last beat with two silences '-'.

At this stage you might start figuring out that BP2 is smart in dealing with rhythmic problems. Imagine doing the previous examples on a MIDI sequencer using event lists or common musical notation... But there is more to come!

What happens if several explicit tempo markers are inserted? For instance,

C5 D5.G5 F5./2 G5 C5 D5./2 D#5 D5 C5.A#4 C5

is equivalent to the preceding example. One of the two markers is redundant. Putting several markers in the same sequence is useless and may even lead to inconsistency. For instance,

C5 D5.G5 F5./2 G5 C5 D5.D#5 D5 C5./3 A#4 C5

results in conflicting durations and will be rejected by the interpreter. A good strategy is to use no more than one explicit tempo marker in an entire musical item, and let the interpreter adjust durations accordingly. This explicit tempo marker is often in the beginning of the item, but it may be inserted elsewhere, notably in case the beginning tempo cannot be specified by an integer number.

.i.**Silences**; (notated '-') may of course be inserted in sequences and will be resized like simple notes or sound-objects, for instance:

C5.D5.- - F5 G5.C5

A sequence of several silences '-' may be replaced with an integer number. Thus, the preceding notation is equivalent to:

C5.D5.2 F5 G5.C5

More generally, silences may be defined as integer ratios. For instance, the item

C5.D5.3/2 F5 G5.C5

is interpreted as:

/7 C5\_ \_ \_ \_ \_ \_ D5\_ \_ \_ \_ \_ \_ - \_ \_ F5\_ G5\_ C5\_ \_ \_ \_ \_ \_

Sometimes it is difficult to calculate the duration of a silence, or it is more convenient to leave it unspecified so that the interpreter determines its value it in the current context. Silences with unspecified durations are called .i.**undetermined rest;s**, notated '.i.\_rest;' or '...'. Try the following

C5.D5.G5..../5 F5 G5 C5.D5 D#5.D5 C5.A#4.C5

or equivalently:

C5.D5.G5. \_rest/5 F5 G5 C5.D5 D#5.D5 C5.A#4.C5

Given the context, the interpreter estimates that the simplest value for the undetermined rest will be 2/5 of a beat. The resulting interpretation is:

/1 C5 D5 G5/5 - - F5 G5 C5/2 D5 D#5 D5 C5/1 A#4 C5

Data file ".i.-da.ShowPeriods;" contains a few examples demonstrating the period notation.

Let's finish with a quizz: can you write a piece of music which keeps accelerating, using a unique symbol for the duration? The answer is 'yes', and the solution is given §4.11 of the Reference Manual.

.i.1.12.2 Polymetric expression;s

All features presented in §1.12.1 apply to .i.**polymetric expression**;**s**, which contain simultaneous musical fragments. The notation{A, B,...} means that expressions A, B,..., are performed simultaneously with identical (symbolic)

durations. Expressions separated by commas are called **fields** of the polymetric expression.

The condition on equal durations is similar to the one in sequence operations. In fact, the same .i.**polymetric expansion algorithm**; is used. For example, the sequence

/1 C4 D4 E4 F4 G4.C5 D5 E5

is interpreted:

/3 C4\_ \_ D4\_ \_ E4\_ \_ F4\_ \_ G4\_ \_ C5\_ \_ \_ \_ D5\_ \_ \_ \_ E5\_ \_ \_ \_

Similarly, the .i.polymetric expression

{C4 D4 E4 F4 G4, C5 D5 E5}

leads to a two-line score represented in a table, the .i.**phase diagram**;:



Note that a .i.polymetric expression; may contain several levels of curled brackets {} and any its fields may in turn be notated in .i.period notation;. Other features such as explicit tempo markers and undetermined rests apply in the same way shown with period notation.

Typical examples of rhythmic items combining period notation with .i.polymetric expression;s are given in the data file ".i.-da.checkPoly;". Simple examples will also be introduced further in the present document.

Read more about .i.polymetric expression;s in the reference manual §4.10.

**2. Sound-objects**

**2.1 Recording a .i.sound-object**

Restart BP2 or, at least, type cmd-n or select "New project" in the "File" menu in order to delete any previously loaded alphabet and grammar and reset all settings to default. Type cmd-h or select "Alphabet" in the "Windows" menu. So far we have not used any alphabet because the first experiment was on .i.**simple note;s** using a predefined alphabet, i.e. C, D, E... and octave numbers in English convention. Let us now create a very simple alphabet of .i.**sound-object;s**. Clear the window and type:

a b c

(You may save it as "-ho.MyAlphabet" if it is too much work!) Now type cmd-e ('edit') or select "Edit prototypes" in the "Windows" menu. This takes you to the .i.**sound-object prototype editor**;. BP2 asks you whether you want load an existing file. Answer 'No'.

The main dialog in the .i.sound-object prototype editor; is shown Fig.2. (Here you'll be in trouble if your screen is smaller than 640 x 480!)



**Fig.2 The** .i.**sound-object prototype editor**; **(main dialog)**

Left and right arrows or the "Go to..." button are used to display other sound-object prototypes.

We will now create a .i.sound-object prototype; named 'a'. Click "Edit/record MIDI prototype". A new dialog appears. Click "Record". Ticks are played to the MIDI output at metronome speed mm = 60. Play a short phrase on the MIDI keyboard and click the mouse. The sound-object prototype appears on the graphic.


**Fig.3 A** .i.**sound-object prototype**; **has been recorded**

Vertical lines in the rectangle indicate MIDI events. If the pitch bender was moved or volume and channel pressure were changed many lines will be visible.

A red vertical line indicates the .i.**insertion point**;. It shows a location that has been clicked on the graphic. This insertion point may be adjusted precisely using left/right or up/down arrows with the command key down.

To hear the prototype *from the insertion point* click "Play" in the same dialog, or press the command key and space bar together. The "Play" button plays the entire prototype only if no insertion point is visible.

Since trailing silences in the beginning and at the end of a .i.sound-object prototype; cannot be recorded, you may want to insert or append them afterwards. "Append silence" prolongates the prototype with a silence. This is

sometimes more convenient than using ordinary silences (notated"-") because, for instance, this prolonged part may be "protected" if property "Never cover end" has been set. If the object is a bell sound, the silence could for instance allow its resonance to be heard fully. In other words, a sound-object may not terminate on the last MIDI message whenever the sound "goes on", or it is to be followed by a certain amount of silence. Similarly, you may want a silence to precede the sound-object, perhaps to enhance its impact. Click ".i.Insert silence;". (Note that if an .i.insertion point; is marked insertion will be attempted at the specified point.)

All buttons in the .i.sound-object prototype editor; are documented on-line: type cmd-? and click a button to get related information. Don't hesitate to play around with these editing procedures. Also try to modify ticks (saved along with the "-mi.<name> " file) and the value of 'tref' which determines the .i.**reference tempo**; of a .i.sound-object prototype;.

Click "Duration/pivot/location" to modify properties of the sound-object prototype. Change the .i.**pivot location**; and watch the effect on the graphic display. The .i.pivot; is used for placing the object on .i.**time streak;s**, e.g. metronome beats. If you inserted a silence in the beginning of the sound-object it may be a good idea to locate its pivot on the first NoteOn. If you are using plucked or drum sounds on a MIDI sampler you may enter precise locations forcing pivots to coincide with the climax of each object rather than the beginning of the sample.

Users of old versions of BP2 will notice that .i.sound-object properties; have been refined: properties such as "OK Rescale", "Expand at will", etc., are still there with the same meanings. But now it is possible to define a range of acceptable .i.dilation ratio;s. As to location, rather than stating that the sound-object is relocatable or unrelocatable, it is possible to define its maximum acceptable shifts either in relative (a percentage of its duration) or absolute value (milliseconds).

If BP2 is controlling a synthesiser it is generally convenient to define sound-objects with property "OK Rescale": MIDI messages contained in the prototype will be spaced according to the metronome value and the .i.**symbolic duration**; of the corresponding sound-object (see §4.2). On the other hand, if BP2 is connected to a MIDI sampler then this rescaling might better be disabled. Since the rich musician's environment often comprises several synths and samplers on various MIDI channels, it is convenient to set these properties separately for each sound-object prototype.

In the same context (several MIDI devices) it is also necessary to declare the MIDI channel of each sound-object prototype: click "Channel/Instrument" in the main dialog (see Fig.2). Changing the channel is self-explanatory.

**2.2 Putting together sound-objects**

Record three prototypes: 'a', 'b', 'c', and save the prototype file as "-mi.MyFile". Now go back to the main dialog (see Fig.2). The "Play object alone" button is similar to the one you have already used, except that it allows you to try dilation ratios -- a good way of determining acceptable ranges.

Type "abcbaa" in replacement of "[Any expression]" and click "Play expression:". You hear the musical item notated "abcbaa". The corresponding

graphic score is displayed on the "Graphic" window if "Show graphic" is checked on the .i."Settings" dialog; (accessed by typing cmd-option space). If you modify pivot positions for 'a' you will quickly figure out pivots. Changing the metronome value will also demonstrate dilation and contraction within the limits that have been specified.

Make prototype 'a' longer than one second and play the expression with mm = 60. The end of 'a' will overlap the beginning of the first occurrence of 'b'. Then set property "Never cover beginning" for 'b' and play the same expression. You will notice that 'b' has been delayed by just the right quantity to prevent overlapping, as shown in Fig.4. This has been achieved by the .i.**time setting algorithm**;.



**Fig.4 An example of score for "abcbaa" in which the two
occurrences of 'b' have been displaced to avoid being
overlapped by preceding objects**

Now, declare that 'b' should **not** be relocated. (Click "Duration/pivot/location" to display this property.) Did you notice that the pivot of 'b' has become a complete arrow? Click again "Play expression". The time setting algorithm will try to "break the tempo" on 'a', thereby meaning that the metronome stops counting until 'a' is over. (This is called *organum* in Western music.) Note the delay on .i.**time streak**;**s** (the blue vertical lines) on Fig.5.



**Fig.5 An *organum* (break tempo) on the same item:
time streaks 1 to 7 have been delayed.**

If you further declare that 'a' should never break tempo, the solution will be that the first 'a' is shifted to the left (forwarded)... Try more! In the end, if 'a' is also not relocatable, no solution will be found: BP2 will take your permission to ignore constraints and produce a score identical to the initial one.

You may refer to §5 of the reference manual for details about the time setting of sound-objects which was demonstrated here. An example is provided with ".i.-da.checkTimeSet;". (Load the file to the "Data" window. This will also load settings "-se.checkTimeSet" which force the time setting process to be displayed.) Display options may be changed by typing cmd-option space or selecting "Settings" in the "Windows" menu.

**2.3 Other ways of creating sound-object prototypes**

2.3.1 Importing a MIDI file

This process is self-explanatory: click the button ".i.Import MIDI file;" on the dialog shown Fig.2 to import the content of a file to the MIDI stream of a sound-object. If you are using Csound the next comprehensive step would be to click ".i.Convert to Csound;" (see §17.8).

Although BP2 recognises type 0, 1 and 2 MIDI file formats, it expects the file to contain a single musical item. Consequently, it will display a warning message if requested to import from a type-2 file containing more than 2 tracks.

.i.2.3.2 Capturing from text

Any ".i.**BP2 score**;", i.e. a string of symbols that BP2 is able to interpret as a musical item, may be captured as a MIDI stream and then pasted to a sound-object prototype. Examples are shown in the data file ".i.-da.makePrototypes;".

Select the text score and choose ".i.Capture selection as...;" in the "Action" menu, or click the button with the same label on the Control panel. If the MIDI output is active the item is played while it is being captured.

The captured content may then be pasted to a sound-object prototype. If the selection point (the vertical red line) is active, several options are offered:

• "Delete and replace" erases the MIDI sequence of the sound-object prototype and replaces it with the MIDI stream.

• "Insert at insert point" inserts the MIDI stream at the insert point marked by a vertical red line on the sound-object prototype.

• "Replace from insert point" replaces part of the MIDI sequence of the sound-object prototype, starting at the insert point marked by a vertical red line, with the MIDI stream.

• "Merge from insert point" merges MIDI messages in the MIDI stream with the ones contained in the sound-object prototype, starting at the insert point marked by a vertical red line.

• "Insert before" inserts the MIDI stream before the MIDI sequence of the sound-object prototype.

• "Append" appends the MIDI stream after the MIDI sequence of the sound-object prototype.

2.3.3 Capturing from a tick cycle

A tick cycle programmed in the .i."Time base" dialog; (see §4.3) may be captured as a MIDI stream. Click button "Capture" in the "Time base" dialog (see Fig.18) and paste the result as indicated §2.3.1.

Note that a new sound-object prototype captured from a tick cycle is automatically declared **periodical** (see §2.5)..i.Periodical sound-object

**2.4 Sound-objects, simple notes, homomorphisms**

.i.Simple notes; are predefined sound-objects containing only a NoteOn/NoteOff pair of messages in the MIDI environment. BP2 allows the combined use of simple notes and sound-objects in the same project.

If the Csound output is active, simple notes are converted to Csound events using the current instrument specifications and default channel-to-instrument mappings (see §17.3).

The labels of simple notes and sound-objects are .i.**terminal symbol**;**s** of the grammar. Whereas the labels of simple notes are predefined, the names of sound-objects must be declared in the .i."Alphabet" window; (which is saved as a "-ho.<filename> " file).

The alphabet may also be used to define mappings of terminal symbols -- both sound-objects and simple notes. Project ".i.-gr.tryhomomorphism;" is a simple grammar illustrating the use of a such a mapping -- more precisely, a .i.**homomorphism**;.

The grammar reads:

 S --> a b c (=X) \* (:X)

 X --> do4 c mi4 fa4

The use of brackets (= )(: ) and the asterisk will be explained in a minute. Let us first look at the ".i.-ho.tryhomomorphism;" alphabet:

-mi.abc

\*

a --> b

do4 --> re4

c --> fa4 --> d

sync a' b' c' d' e e' f f' g g' h h' i i' j j' k k' l l' m m' n n' o o' p p' q q' r r' s s' t t' u u' v v' w w' x x' y y' z z'

Sound-objects are named 'a', 'b', 'c', 'd', 'sync', etc. Their detailed specifications are found in the file "-mi.abc". The asterisk is not a terminal symbol. BP2 recognises it as the label of an .i.**homomorphism**;. It does so because it finds arrows in the following list of terminal symbols. Thus, an expression like

a --> b

means that 'b' is the image of 'a' through the homomorphism notated '\*'. We will see later how this mapping is used. Similarly, the alphabet indicates that 're4'(a simple note) is the image of 'do4', 'fa4'the image of 'c', and 'd' the image of 'fa4'. Sound-objects can be mapped to simple notes and conversely.

Every terminal symbol (sound-object or simple note) that is not explicitly mapped to another one is mapped to itself. Thus, the image of 'b' is 'b', etc.

Since this mapping is a .i.**homomorphism**;, strings of terminals are also mapped in a straightforward manner. Thus, "b re4" is the image of "a do4" since 'b' is the image of 'a' and 're4'the image of 'do4'. To complete the scene, the image of an empty string is an empty string.

Several homomorphisms may be defined on the same alphabet. Their descriptions should be separated with lines of hyphens, and the labels of homomorphisms should appear on top of each section. If a section contains at least an arrow, BP2 understands that the first line contains the label of a homomorphism, otherwise it will pick it up as a new terminal symbol.

Let us now see how this simple .i.**homomorphism** ;is used.

When running the grammar (cmd-r) a single musical item is produced

a b c (= do4 c mi4 fa4) \* (: re4 c mi4 d)

which is actually printed without brackets:

a b c do4 c mi4 fa4 re4 c mi4 d

The leftmost bracket (marked with '=') is a 'master' expression while the rightmost one (marked with ':') is its 'slave'. The slave bracket reproduces exactly the content of the master bracket it is attached to. Exactly? Not here: since the slave bracket is preceded by '\*' its content is replaced with a mapping of the master bracket through homomorphism '\*'.

You do not need to understand everything about homomorphisms to go on with BP2. (More information is available in the reference manual §4.1, and publications, notably Kippen & Bel 1992.) The idea came from the need of a general syntactic model coping with repetitions and pseudo-repetitions in music. The first homomorphism we ever used was a transformation known to North Indian tabla players: when repeating patterns they often change voiced sounds such as 'dha', 'ge' to their unvoiced counterparts 'ta', 'ke'... (See ".i.-gr.dhin--;" for instance.)

Projects ".i.-gr.Ruwet;" and ".i.-gr.cloches1;" are illustrations of homomorphisms used to modify tonal and modal patterns.

**2.5 Cyclic sound-objects**

Any sound-object may contain a periodical part starting at an arbitrary point of its time-span interval. Clicking the "Period (cyclic object)/strike mode" button on the dialog shown Fig.2 displays a dialog for declaring periodicity.

**Cyclic sound-objects** are handled in a particular way: if their dilation ratio in the performance (see §3.1) is larger than 2, they are repeated instead of being dilated. An example is given with ".i.-da.preroll;". The item

{C6 B5 A5 G5 A5 B5 C6, cycle1}

contains 'cycle1', a sound-object which has a periodical part in the last 60% of its time-span interval. The resulting time setting is shown Fig.6. The non-periodical part (40% of the duration) is first performed, then the periodical part is repeated 7 times in synchronisation with the sequence of simple notes.



**Fig.6 A performance with a cyclic sound-object**

Note that the pivot of 'cycle1'is set at the beginning of its periodical part to ensure its proper location.

There are two typical cases of periodicity, both handled by BP2:

• The sequence of events in the periodical part must be sent again to the sound device when that part is repeated. This happens for instance with sound-objects handled by a MIDI synthesizer.

• The object is declared cyclic but there is no need to repeat the sequence in the periodical part. This happens with sound-objects handled by a MIDI sampler. The sampler expects BP2 to trigger the process with a NoteOn and then it takes care of repetitions until a NoteOff is received.

In the latter case, property ".i.Discard NoteOff's except in last period;" should be activated. It prevents the sampler from setting off as soon as the first cycle has been completed. This property is generally completed with ".i.Don't strike again NoteOn's;" accessed in the same dialog and documented on-line.

Property ".i.Force integer number of periods;" adjusts the durations of sound-objects to make sure that the last repetition of the periodical part will be complete. It is only required in case the acceptable .i.dilation ratio; of the sound-object (see reference manual §3.1) is less than 2.

**2.6 Pre-roll and post-roll**

These words are familiar to folks working on sound or video editing. On a tape recorder in pause-recording mode, the .i.pre-roll; is the delay between the moment the pause button is released and the moment the recording actually starts. Similarly, the .i.post-roll; is the delay between the time the pause button is pushed and the one it actually stops recording. With automatic editing devices such as VideoDirector, these mechanical parameters may be entered in the program's settings so that the program compensates them and achieves accurate timings.

2.6.1 Pre-roll

In BP2, the .i.**pre-roll**; of a sound-object is the delay between its first event (Csound event or MIDI message) and the beginning of its .i.time-span interval;. **The first event is the one that tells the sound processor to start**, whereas **the beginning of the time-span interval is the moment when the listener feels it actually started**.

A positive pre-roll allows some events to fall before the .i.on-setting time; of the sound-object. This is useful when the sound-object contains an initialisation sequence that should not be accounted in its time-span interval.

Fig.7 shows a sound-object labelled 'un' with 482ms of fixed pre-roll.



**Fig.7 A sound-object with positive pre-roll**

On the graphic (right part of Fig.7), events are shown as short vertical lines. Four events are located outside to the left of the time-span interval.

2.6.3 Post-roll

In BP2, the .i.**post-roll**; is the delay between the last MIDI message of a sound-object and the end of its .i.time-span interval;.

A negative post-roll allows some events to fall beyond the off-setting time of a sound-object. This may be useful if the sound-object contains a final sequence of events that should not be part of its time-span interval.

A positive post-roll is the proper way of compensating a mechanical delay in setting-off the sound device. If for instance the event sequence in the sound-object ends with a NoteOff and the device takes 50ms to react, then setting the post-roll to 50ms will append a 50ms silence after the sound-object to allow the device to finish its job. This is generally combined with a positive pre-roll compensating on-setting delays.

2.6.4 An important formula

.i.Sound-object duration

 = date of last event - date of first event - pre-roll + post-roll

**3. Opening and compiling a grammar**

If you are in a hurry to hear "Turkish" music you may jump to §5.2 because BP2 automatically compiles the alphabet and grammar when prompted to produce items...

**3.1 What's in the examples folder?**

Folder "BP2 examples" contains the following types of files:

• .i.-gr.<name> ; is a file containing a grammar (or several subgrammars, see reference manual §4.4).

• .i.-ho.<name> ; is a file containing the names of sound-objects, i.e. the terminal alphabet of a grammar, plus (if required) the information about their possible homomorphic transformations (.i.homomorphism;s). (See §2.4 or reference manual §4.1)

• .i.-mi.<name> ; is a file containing MIDI code defining sound-objects, along with their metrical or topological properties (see §2.1 or reference manual §3).

• .i.-da.<name> ; is a file containing (textual) productions of the grammar (or any musically significant string of terminals/variables).

• .i.-in.<name> ; contains data relevant to real-time interactive control (see §6).

• .i.-se.<name> ; is a file containing settings for a project (see §16).

• .i.-kb.<name> ; is a file containing a keyboard encoding (see §10.4).

• .i.-wg.<name> ; is a file containing rule weights (see §5.2, §5.4, reference manual §4.6).

• .i.-gl.<name> ; is a file containing a glossary (see §7).

• .i.-tb.<name> ; is a file containing a time-base (see §3.6.3).

• .i.-cs.<name> ; is a file containing the description of Csound instruments (see §17).

• .i.or.<name> ; is a file containing a MIDI orchestra (a set of patches assigned to MIDI channels).

• .i.+ sc.<name> ; is a file containing a script (see §8).

All these files are in 'TEXT' format but they may have different .i.file types; to facilitate their selection in BP2's load/save operations. These types are documented on-line: click ".i.Types-creators;" in the "Frequently Asked Questions" dialog.

BP2 files can be loaded directly in word processors. Once the file has been edited save it as "ASCII" or "pure text". Then retrieve it under BP2 using the "load any file" option: type cmd-o with the option key down.

**3.2 Headers and related files**

Quit BP2 and double-click ".i.-gr.koto3;". This will automatically load the grammar and the related "-ho.abc1" alphabet, for two reasons:

• The first line of grammar "-gr.koto3" contains the name of its alphabet.

• ".i.-ho.abc1;" (or an alias of it bearing the same name) was in the same folder as "-gr.koto3".

Here is a very important feature of BP2: "projects" are made of several files linked together. This is a modular approach since that the same file (e.g. "-ho.abc1") may be shared by several projects. Whenever BP2 loads a file it scans its top lines searching the names of related files. If related files are found they will be loaded whenever necessary. For instance, the top of the grammar window for project ".i.-gr.koto3;" indicates:

-or.koto3

-in.abc1

-tb.koto3

-se.koto3

-ho.abc1

i.e. the names of several related files. (Their use will be explained later.) In turn, the "Alphabet" window, in which ".i.--ho.abc1;" has been loaded, contains the name of the sound-object prototype file ".i.-mi.abc1;" required for its operation.

Generally you don't need to type or modify headers. For instance, if the grammar contains no header, as soon as you load an alphabet BP2 will insert a header with its name on tip of the "Grammar" and "Data" windows. If you save the alphabet file under a different name, or open another file, these headers are automatically updated accordingly. If you delete the alphabet headerers are removed...

You can look at the different files loaded: see menu "Windows" and try to remember keyboard shortcuts. As usual with Macintosh® software, BP2 keeps a trace of changes and prompts you to save or discard them before you clear a window or exit the program. However, it has a non-standard an altogether safe feature: the first time you type cmd-s after making changes, it prompts you to confirm the name and location of the file.

Closing a window with text or graphic data does not clear its content, except for the "Trace" window. It is sometimes useful to simplify the screen display. Systems above 7.5.1 also make it possible to reduce windows to their drag regions: a double or triple click on the title bar does the job.

**3.3 Organising projects**

It is important to store related files *or their aliases* in the same folder so that BP2 finds them immediately. Reserve a special folder to BP2 software, on-line documentation and start-up/shutdown files, as shown Fig.8.



**Fig.8 The software folder**

MacOS offers an elegant solution for arranging projects: use **aliases** instead of moving or duplicating files. You may reserve another folder for the **basic elements**: alphabets, sound-object prototype files, glossaries, etc., as shown Fig.9.



**Fig.9 The folder containing basic elements**

File names are just indicative and should in general be more explicit. Names in italics are **aliases**. Here, for instance, an alias "-*ho.alphabet1*" is required in the same folder as "-gr.grammar1" because the name of this alphabet file appears on top of the grammar file. Similarly, other aliases have been placed near the files calling them. An alias of "-se.startup" is needed in the "Settings" folder so that scripts will execute the "New project" instruction properly.

Note that aliases for BP2 must bear exactly the names by which the files are called when BP2 reads headers. Here we used the same names for aliases and original files, the only way to distinguish them being the italic.

It is advisable to create a separate folder for saving your current work, given that whenever you create new basic elements you will continue storing them in the folders shown Fig.9. A typical example of work folder is shown Fig.10.



**Fig.10 A folder for current BP work**

Here, Project 1 deals with a grammar (and associated alphabet, etc., files that are automatically loaded), while Project 2 uses only a data file (and associated files). Data files "-da.data1" and "-da.data2" call alphabet, settings, glossary and interactive file, aliases of which have been placed into the "Data" folder.

Some aliases in the "Scripts" folder are shown between brackets because they are only needed if a script calls them directly. Besides, scripts are capable of recording directory and volume changes (see §8).

Data on the demonstration disk is not organised in the rational way explained above, because aliases loose track of their original files if both are moved to a different volume.

**3.4 Network operation**

Since BP2 is able to resolve aliases and load files on remote volumes, this type of organisation may be extended to local network configurations: the basic elements could for instance be stored on a server and accessed by users with the usual read/write permissions. For instance, it is safe to protect alphabets or sound-objects from unauthorised modifications if they are shared by different projects.

BP2 can read directly text files edited by .i.DOS or Windows editors;. In this process, "high ASCII" characters are automatically transcoded. This feature has been implemented to facilitate the direct interpretation of text-oriented musical items or grammars produced in the Wintel environment.

**3.5 A very simple grammar**

Quit BP2 and double-click ".i.-gr.doeslittle;" in the "BP2 examples" folder. This will start again BP2, bypassing the welcome dialog, and the content of the "Grammar" window will be displayed as follows :

-se.doeslittle

// A very simple grammar that does almost nothing

RND

S --> C4.C5.D5 C5 F#4.A4.C5

S --> G3 C4\_ B3 C4

S --> {C5 D5 E5 F5, G5 D5 C6}

Type cmd-k or select "Compile" in the "Action" menu. BP2 quickly compiles the grammar, i.e. it generates some internal code (**tokens**) that will enable it to produce items.

Now type cmd-r ("run") or select "Produce items" in the "Action" menu. After a short while an item is displayed in the graphic score and played on the MIDI output. You are given the option to produce more, play again, display as text or cancel. When producing more you will notice that the inference engine (the machines that "runs" the grammar) selects any of the three rules in the grammar to produce an item. This selection is random with equal probabilities, as stated by the instruction "RND" on top of the grammar.

The operation of this grammar is very simple : it starts with a .i.**work string**; containing only 'S', the .i.**starting symbol**;**.** A rule is selected at random. The selected rule replaces 'S', its left argument, with its right argument. Then no change is possible any more ; the item is displayed and played.

Type cmd-option space or select "Settings" in the "Windows" menu. A .i.**"Settings" dialog**; in two parts is displayed, along with three more option dialogs. These will be analysed in greater detail (see Fig.24 in §5.2). These settings have been loaded automatically from the file "-se.doeslittle" whose name appears on top of the "Grammar" window. For the time being, just watch the checked items : "Show graphics" forces graphic display, and "Use MIDI in/out" allows items to be played on the MIDI device. You may for instance check .i."Non-stop improvize" button; and type cmd-r again. Now BP2 produces items continuously. To stop the production, hold the mouse button down and then click "STOP" when the red dialog appears in the top-left corner.

Improvizing on a slow computer may look deceiving because BP2 takes time to produce a new item and does not keep the tempo. Check the .i.**"Compute while playing" button**; to allow BP2 to proceed with calculations and keep items ready to play in a queue. If the grammar is not too complex, the tempo will be maintained. The drawback is that when you click to stop the improvisation, items still in the queue may continue playing. Clicking again will abort playing, but it may keep a note hanging because its 'NoteOff'

has not been received. Flush the output clicking the .i.**"MIDI PANIC" button**; on the ".i.Control Panel;".

Every time the grammar is run it produces exactly the same sequence of pieces. This is due to the fact that BP2 resets its inner random number generator unless the .i.**"Allow randomize" button**; is checked on the upper "Settings" dialog.

**3.6 A more elaborated project**

Type cmd-l or select "Load project" in the "File" menu to load a project : you will be prompted to locate a grammar, and related files will be automatically loaded. Before loading a project, BP2 clears all project windows ; the same may be done by typing cmd-n or selecting "New project" in the "File" menu.

Clearing the current project is a good habit when you start a new job. If windows are not cleared they may still contain headers causing unwanted files to be loaded.

Look carefully at the ".i.-gr.koto3;" grammar file. On top of it you read :

-or.koto3

-in.abc1

-tb.koto3

-se.koto3

-ho.abc1

These are the names of the .i.**MIDI orchestra file**;, the .i.**interactive file**;, the .i.**time base file**;, the .i.**settings file**; and the .i.**alphabet file**; respectively. The file "-in.abc1" will be used for controlling BP2 interactively from a MIDI keyboard or controllers (see §6).

The "-se.koto3" settings file has already been automatically loaded. If BP2 does not find its name in the "Grammar" window it searches it in the "Data" window.

.i.loading any file;You can load grammar and alphabet files separately by typing cmd-o or selecting "Open file" in the "File menu" when the corresponding window is active. This additionally enables you to load files of types different of BP2 types, for example a grammar that has been edited in a word processor and saved as pure text: type cmd-o with the 'option' key down.

The "-mi.abc1" file has not been loaded so far. BP2 will open it only at the time it is needed.

**3.7 Tokenised alphabet and grammar**

If you type cmd-h with the 'option' key down you have the option to display the alphabet in its "tokenised" form. A .i.tokenised alphabet/grammar/glossary; is useful to show whatever the compiler has been able to understand. It is displayed in a multi-purpose window called the "Trace" window..i.Compilation;


**Fig.11** .i.**Tokenised alphabet**; **in the** .i.**"Trace" window**;

The tokenised alphabet actually displays the entire mapping of .i.**homomorphism**; notated '\*' that was partly defined in the original alphabet (see §2.4). Predefined symbols '-' and '\_' represent .i.**silence**;**s** and .i.**prolongation**;**s** respectively. These are not modified by any homomorphism. Other symbols are specific to this "-ho.abc1" alphabet. Homomorphism '\*' changes 'a' to 'a'', but 'a'' remains unchanged. Symbol 'chik' is also not affected by the transformation. (See §2.4 or reference manual §4.1 to understand homomorphisms.)

Looking carefully at the alphabet you may notice that a new terminal symbol 'sync' has been appended: it was not in the original file "-ho.abc1". The symbol was actually found between single quotes in the grammar, and therefore appended at compile time. Similarly, .i.out-time sound-object;s between angle brackets <<>> may contain terminal symbols that BP2 creates at compile time.

The tokenised grammar is shown Fig.12.


**Fig.12** .i.**Tokenised grammar**; **in** .i.**"Trace" window**;

All rules have been numbered and contain .i.**weights in grammar;**. (See an explanation of weights in reference manual §4.6) Rules that were not assigned explicit weights appear with default weight <127>, a value that fits well with MIDI parameters.

Also note that commentaries [between square brackets] or lines starting with double slashes have been left out.

.i.Rule numbering; has no real effect on computation. For instance, the order in which rules are scanned in "ORD" grammars (see reference manual §4.4) is always their actual order of occurrence in the grammar. Nevertheless, BP2 renumbers subgrammars and rules at compile time. These numbers are used for .i.conditional jumps; found in .i.**grammar procedure**;**s** (see §8.1 of reference manual).

**3.8 Checking .i.grammar consistency**

Selecting "Check variables" in the "Search" menu produces a report on variables used in the grammar. Some variables may be **undefined**, i.e. they appear in the .i.right argument of a rule; and nowhere in the left argument of a grammar or glossary rule. The grammar might therefore produce final strings containing these variables. BP2 warns the user at compile time if the grammar contains .i.undefined variable;s. The grammar may nevertheless be used: when interpreting the final string as a musical item, BP2 ignores left-over variables. When compiling a glossary, however, an error message is returned if any rule or "Define" instruction produced an undefined variable.

Conversely, .i.**unreachable variable;s** are the ones found in the .i.left argument of a rule; although they appear in no right argument, so that the grammar will never produce them. This is generally not considered a mistake because a grammar may contain unreachable variables coming from an earlier version. Rules containing them will never be candidates. Glossaries may also contain many unreachable variables because a project is generally not using all functionalities of MIDI devices -- it is not likely to make use of the 128 instruments defined in General MIDI, for instance.

When .i.compiling a glossary; (see §7), BP2 makes sure that it does not generate variables appearing in the left argument of a grammar rule. Since the glossary is used only once all candidate rules in the grammar have been exhausted (or there is no grammar), the grammar may not be invoked afterwards.

**3.9 The start string**

Select ".i.Start string;" in the "Windows" menu. A window appears containing the string from which computation will start in production mode:



**Fig.13** .i. **"Start string" window**;

Generally the .i.start string; is predefined symbol 'S', but sometimes you may want to derive other strings of variables/terminal symbols, especially while debugging a grammar. Type or paste start strings on separate lines in the "Start string" window, then select the one you want by highlighting it with the mouse. If you select no start string then the first one in the window is always used. Since 'S' is already there whenever you start the program you may ignore this "Start string" window until you decide to use alternate start strings.

.i.Start string;s are stored in the "-se" settings file (see §16).

Another, more intuitive, way of .i.using any expression as a start string; is to select it in any window (including the grammar itself) and type cmd-p or select "Play selection" in the "Action" window. Normally this command tries to interpret the selection (using the alphabet), but if the selection contains variables then it is taken as a start string and the grammar is prompted to derive a **single** item from it.

**4. Producing items**

There is a window called the .i."Data" window; accessed from the "Windows" menu or by typing cmd-d. We will use it now. In addition, we will display the .i.**Control panel**; (see the "Windows" menu or type cmd-=).



**Fig.14** .i. **"Data" window; and** .i.**Control panel**;**.
Grammar and glossary are visible behind these windows.**

**4.1 Listen to this one!**

Click button ".i.Produce items...;" or type cmd-r. (Old-timers will remember the "Run" command in Basic!). You see messages indicating that BP2 is computing the item. (Don't hurry to read messages: there is a command in the "Windows" menu recalling the latest 30 messages.) Eventually an item will be displayed.

Produce more items in the same way. The sequence shown below is dependent on the random sequence generator, and may not be the same on different machines.


**Fig.15 Four items have been produced by ".i.-gr.koto3**;**"**

These items are in text format, not conventional musical score notation, but BP2 will later manage to convert them to sounds on the basis of information contained in the "-mi.abc1" sound-object file.

While BP2 is at work it may be interrupted by keeping the mouse button down. A small window appears with ".i.Resume; button" and ".i.Stop; button" buttons allowing to continue the job or abort it. "Resume" and "Stop" commands appear in the "Action" menu as well. You may also type cmd-r or cmd-<period>. During the interruption you are allowed to do almost anything, for example edit the alphabet and grammar, change options, etc. This may be necessary if the computation has become too slow and you want to understand why it is taking such a long time: select for instance "Step-by-step produce" and then click "Resume" or type cmd-r . (Note that options are related, therefore if you

select "Step-by-step produce" then "Display production" will also be selected.) If the computation is looping you may modify the subgrammar to find your way out. Understandably, BP2 might decide to recompile the current grammar before resuming computation.

Because of this possibility of "doing almost anything" while a process is interrupted, you may sometimes enter into unpredicted situations in which BP2 is not able to resume the process properly. Many unusual situations have already been encountered and are handled by BP2, but it is virtually impossible to check all possible combinations. Therefore if you are in the course of an important process it is safe to start afresh before making radical changes.

Let us assume that a few items have been produced in the "Data" window, as shown on Fig.15. Each item is actually displayed as a single line, more precisely a "paragraph" terminated with a carriage return. You may of course type items yourself if you know the alphabet of terminal symbols recognised by the "-mi." sound-object file.

For the next experiments it will be more practical to use the Control panel (see Fig.14) than to pull down menus. Type cmd-= or select "Control panel" in the "Windows" menu.

Select one or several item(s) with the mouse. Click button "Play selection" or type cmd-p. First, BP2 loads the "-mi." .i.**sound-object file**; whose name is specified on top of the "Alphabet" window. The "-mi.abc1" file is loaded forever except if you make changes in the alphabet, in which case BP2 must look for definitions of new sound-objects which the modified alphabet might be refering to.

After a short while you see the .i.**sound-object structure**; (the musical item) displayed in the graphic window while the resulting sound is played on the MIDI output.

Below is the graphic display of item

\_vel(127) \_volume(40) b b <<f>> {5,a c b,f - f} a <<chik>> b <<sync>>

which was produced by this grammar.



**Fig.16 Graphic representation of** .i.**sound-object structure;:**\_vel(127) \_volume(40) b b <<f>> {5,a c b,f - f} a <<chik>> b <<sync>>

This graphic is a kind of musical score with objects represented as rectangles. Background blue lines are .i.**streak**;**s** of the .i.**time structure**; and red arrows mark the .i.**pivot**;**s** of sound-objects. (See §2.1 or §3 of the reference manual; read Bel 1992a for a detailed presentation) A tail-less arrow denotes a .i.**relocatable object**;, i.e. one whose pivot may not coincide with a time streak. If a sound-object is truncated the deleted part is displayed with dotted lines.

Simple notes (see §1.3, §1.11) are also represented with rectangles and with no pivot since they are relocatable.

Note that you can modify colours by selecting "Change colours" in the "Layout" menu, and save the new options along with "-se.koto3" settings. Again, users of black and white monitors should uncheck the "Use color" option in the same menu.

A scale of physical time appears on top of the window. Here the tempo was 400 beats per minute, therefore 400/60 = 6.66 beats per second.

You may change the graphic scale of graphics by selecting ".i.Graphic scale;" in the "Misc" menu. You must play the item again so that the effect of colour or graphic scale changes become visible.

**4.2 What's in a .i.polymetric expression;?**

May be you liked the way this item sounded (on a "Koto" patch of the D-50, or vibraphone in QuickTime music) and want to understand its time structure. Click the "Data" window (or type cmd-d) and select the item with the mouse. Then click button ".i.Expand display ;". You get this:



**Fig.17 Expanding a** .i.**polymetric expression**;

Expressions with curled brackets {} and commas are called .i.**polymetric expression**;**s** and represent polyphonic .i.sound-object structure;s. These expressions are incomplete descriptions of the actual precedence relationships between sound-objects and/or notes in the structure. BP2 is able to construct a .i.**complete polymetric expression**; by adapting tempos in superimposed sequences. (This process is explained in great detail in Bel 1990a-b,1991,1992.) The expanded display shown Fig.17 is an outcome of the expansion algorithm.

BP2 expands *any* .i.polymetric expression; (including a simple sequence) before making sounds. Once the expression has been expanded, the following score-like .i.**phase diagram**; is constructed:

in which it is clear, for instance, that the first occurrence of 'f' begins with the first occurrence of 'a'.

Symbols '\_' prolongate sound-objects or notes. For instance, the .i.**symbolic duration**; of the first occurrence of 'b' is three time units whereas the last occurrence of 'f' is five time units. (In this particular example, a time unit is one third of a beat.)

Symbols <<f>>, <<chik>> and <<sync>> represent .i.**out-time sound-object**;**s**, i.e. objects with no duration. Here for instance (from the information found in the "-mi.abc1" file), <<f>> will be a NoteOn on key F4 followed immediately with a NoteOff. <<chik>> is a chord plucked in the same way. It is only audible if the NoteOff instruction does not suppress sound immediately. Therefore we used a "plucked string" patch.

Out-time sound-object <<sync>> produces a C5 on MIDI channel 15 that does not make any sound but may be used to synchronise another BP2 running on the same MIDI network (see §6.4).

**Performance controls** .i.performance control;\_vel(127) and \_volume(40) are attached to the next object or silence appearing in the same sequence, and are not displayed on graphics. Remember that you can attach very subtle variations of all MIDI parameters, notably pitchbend, modulation, aftertouch, to any structure of sound-object or/and simple notes. (See §1.4-10 regarding performance controls.)

Many .i.polymetric expression;s are found in project ".i.-gr.polyphony1;".

You may select only part of the item (even a single terminal symbol) and listen to the way it sounds (type cmd-p). Try to figure out the difference between "striated" and "smooth" time by carefully looking at the graphics these different settings generate. (See §9 regarding .i.smooth time;.)

**4.3 The .i."Time base" dialog**

To change the .i.metronome value;, type cmd-t. The "**Time base**" and "**Metronome**" dialogs are displayed (see Fig.18).



**Fig.18 "Time base"and "Metronom" dialogs**

There are two ways of adjusting the clock. The .i."**Metronom**" dialog; deals with conventional metronome settings, i.e. "mm = 400" means 400 ticks per minute. You may change this value, for instance, to 345.08. Changes are reflected in the .i."**Time base**" dialog;. Now it tells "8627 ticks in 1500 seconds", which actually means the same as "mm = 345.08". If you need to set the time base accurately, type directly the numbers of ticks and/or seconds. Whenever you change values, BP2 finds the simplest integer numbers yielding the expected integer ratio. The "Metronom" dialog may display a rounded value which will not affect accuracy since BP2 actually uses the integer ratio displayed on the "Time base" dialog.

A very accurate clock is needed to superimpose BP2 items on a given (digitized) sound track. In addition, synchronisation tags (see §6.6) or Apple Events (§6.7) make it possible to start items on precise dates determined by the environment.

The .i."Time base" dialog; contains boxes allowing the definition of **three independent cycles** that produce .i.**ticks**; on marked beats. When ".i.Play ticks;" is checked, ticks become audible. Ticks may be mapped to any MIDI key, channel and velocity. For each cycle, first adjust the cycle duration (1 to 40 beats), then check the boxes of all beats that need to be stressed by a tick. Each cycle may run at a speed differing from that of the basic clock if its speed ratio is not 1/1. A typical example is given in the time base saved as ".i.-tb.slowshift;" which may be opened (cmd-o) when the .i."Time base" dialog; is in front.

It is possible to define each tick of the time base individually. Click any square of a tick cycle with the 'option' key down to display the .i."**Tick settings**" dialog; (Fig.19).



**Fig.19** .i.**"Tick settings" dialog**;

The ON/OFF button determines whether the beat should be audible. If "Use default settings" is checked then the tick will be heard on the default channel, key and velocity of its cycle. If "Use special settings" is checked, then these parameters are specified individually as shown Fig.19. The ".i.Capture settings;" button makes it possible to pick up these parameters from a MIDI device. It's a good idea to send ticks to a .i.MIDI sampler; with percussive or fancy sounds.

A tick cycle may be captured as a MIDI stream and pasted to a sound-object prototype (see §2.3.3). Ultimately, if the sound-object is played alone it can further be saved to a .i.MIDI file; (see §14.1). This is a good trick to use tick cycles created by BP2 with other MIDI software.

The time base continues ticking, if instructed to do so, while BP2 is improvising items. You may check this with examples contained in ".i.-da.tryTicks;". Note that in .i.**smooth time**; (see §9) beats may become irregular, depending on time streaks created by the musical item. Try ".i.-gr.tryTimePatterns;" to hear the difference. These superimposed ticks are neither saved in MIDI files nor interpreted as Csound events.

Different settings of the time base have been saved as .i.-tb.<name>; files representing the most popular *tala*s of North Indian classical music.

The time base allows an .i.**accurate transcription of durations**; when notes are entered from the MIDI keyboard (typing cmd-j, see §1.5). When ".i.Play ticks;" is checked, durations are transcribed following the .i.**period notation**; (see §1.12.1).

**4.4 Blowing memory on fractals...**

Instead of listening to a selected item, you may copy and paste parts of it. For instance, select the whole item shown on Fig.16, type cmd-c or select "Copy" in the "Edit" menu, then select the first occurrence of 'a' and type cmd-v or select "Paste". In this way, you replace 'a' with the whole item "b b <<f>> {5, a c b, f - f} ...". The resulting performance is the following:



**Fig.20 A "fractal" musical item**

You can hear (and see) that the overall duration of the item is unchanged although it has become more complex. Now, time units (for the desired accuracy) have become so small that streaks appear as a grey background. Fractal fanatics may repeat the operation several times or figure out a self-imbedding grammar doing it automatically. Indeed, you'll end up blowing the computer's memory, so be careful to save windows before producing items and try to set a computation limit using .i.**dynamic weight;s** (§4.6 of reference manual), flags in programmed grammars (§11 infra) or limited buffer size (§4.7 infra). Quantization (see §6 of reference manual) is also strongly recommended in this context.

You can type cmd-p while a string containing terminal symbols and variables is selected in any window. For instance, bring to front the alphabet window and listen to sound-objects 'a', 'b', 'c', etc., separately. Then bring the grammar window to front and select any string containing terminal symbols. If variables are found, the selection is taken as a start string in production mode.

**4.5 Victor Hugo against Darius Milhaud**

You will appreciate BP2's ability to produce intricate .i.**polyrhythmic pattern;s** by playing item produced by ".i.-gr.Djinns;" with percussive sounds (e.g. Roland's "C-54 percussion set 1"). The piece was created to synchronise the recitation of Victor Hugo's poem "*Les Djinns*" with the opening section of "*La Création du Monde*" by Darius Milhaud (EMI CDC-7 47845 2) in Andréine

Bel's choreographic work for *Collège des Prêcheurs* and *Conservatoire Darius Milhaud* (Aix-en-Provence, 1992). We found that the meaning of the poem would coincide with particular events in the musical piece. Then we calculated a time-base value fitting the two tracks together: setting 1000 beats in exactly 1578 seconds was possible since BP2 does not round integer ratios.

Poem "*Les Djinns*" is built on verses with variable meters: 2/2/2/2/3/3/3/3/4/4/4/4 ...up to 10/10/10/10 --except 9/9/9/9-- and then decreasing (see appendix).

We decided that every verse should have the same duration marked by 16 beats with a stronger first beat. The result is a polyrhythmic piece with one instrument showing beats at a fixed speed and another one playing ticks at variable speeds. Listening to variable parts made it possible for children to recite the poem at the expected tempi.

The grammar below shows how the piece was produced. It has 15 sections labelled "A2", "A3", etc. Each section puts together a fixed motive "M" and a variable motive "T16", "T24", etc. Here, "T16" stands for 16 ticks played as note "do4". Fixed motive "M" is a string of alternated "do2" and "do3" with a stronger beat in the beginning, where "sib5" is played together with "do2".

-se.Djinns

ORD

S --> A2 A3 A4 A5 A6 A7 A8 A10 A8 A7 A6 A5 A4 A3 A2

-------------------------

SUB1

A2 --> {M,T16}

A3 --> {M,T24}

A4 --> {M,T32}

A5 --> {M,T40}

A6 --> {M,T48}

A7 --> {M,T56}

A8 --> {M,T64}

A10 --> {M,T80}

-------------------------

ORD

T16 --> T8 T8

T24 --> T16 T8

T32 --> T16 T16

T40 --> T32 T8

T48 --> T32 T16

T56 --> T48 T8

T64 --> T32 T32

T80 --> T40 T40

T8 --> Tik Tik Tik Tik Tik Tik Tik Tik

-------------------------

SUB1

M --> {1,do2,sib5} do3 do2 do3 do2 do3 do2 do3 do2 do3 do2 do3 do2 do3 do2 do3

Tik --> do4

**Fig.21 A grammar for reciting poem "Les Djinns"**

Below is a graphic example of the output of "A6 A7".


**Fig.22 The "A6 A7" tick pattern in "Les Djinns"**

If only variable parts must be heard then regular beats "do2" and "do3" may be replaced with silences.

It is easy to figure out that the full piece would require an excessively large .i.phase diagram; (more than 200,000 columns) because it is based on the lowest common multiple of 2, 3, 4, ... 10. Therefore it can played only if .i.**quantization**; is set to a reasonable value (see §6 of reference manual). We found that musicians were be satisfied with a 50 millisecond quantization even though deviations of ±50ms were still noticeable in the "A7" part. With this setting, computation requires a 2 Mbyte memory partition for BP2 and it takes a couple of minutes on a Mac IIci, so be patient if you can't borrow your rich neighbour's PowerMac...

You may use almost the same grammar with syllabi of the poem stored on a sampler and triggered by appropriate keys. I prefer live children's voices...

**4.6 Graphic scale and time accuracy**

Fig.23 shows a few dialogs activated by the "Misc" menu.



**Fig.23 Graphic scale, time accuracy, etc.**

.i.Time accuracy; is set in the "Accuracy" dialog. .i.**Time resolution**; is the shortest significant time interval, i.e. the resolution of the in-built MIDI driver. Its default value is 10 milliseconds, but you might want to change it (down to 1 millisecond). Because of the inaccuracy of MIDI devices, however, time resolution less than 10ms is rather illusory.

Note that timings in the Csound output (see §17) are not influenced by the time resolution.

Internal computations use 1ms resolution. BP2's clock uses *long integer* numbers in range 0..231. This allows you to handle items lasting up to:

0.001 x 231 = 2,147,483 seconds = about 24 days...

The "Accuracy" dialog also displays the setting of .i.**quantization**; (see reference manual §6). ".i.**MIDI set-up time**;" is an estimation of the time needed by MIDI codes to transit through the MIDI driver. A 100ms value is safe. Small values sometimes lead to incorrect timings in the beginning of items.

**4.7 Buffer size**

Normally BP2 takes care of the size of the .i.**work string**; (the item being computed) and expands its work space when necessary. You may want to control it, notably when using a length-increasing grammar. Dialog "Buffer size" (see Fig.23) displays the current buffer size and the default one assigned when computation starts. Suppose that you do not want to produce a string longer than 750 symbols. Type "750" in the upper field and any *equal or smaller* value in the default field, then select "use". Now BP2 computes items until the length of the work string reaches 750 symbols, then it prompts you to decide whether you want to stop or continue (expanding the current work space).

In the ".i.-gr.Mozart;" project (see §5.4 infra), default buffer size is set to 500 symbols but all items need more space. Therefore, BP2 is forced to expand its space for the work string the first time it produces an item. It does so without asking any permission because the ".i.Don't stop;" option is on. The new value is stored into dialog ".i.Buffer size;". (Buffer expansion is done stepwise: size is recurrently multiplied by 3/2.)

Setting a very high default value reduces the memory available for other operations. Setting a too small one reduces speed during the first computation, as BP2 needs to resize the work string buffer more often.

A typical application of buffer size limitation is project ".i.-gr.cloches1;", an extension of project ".i.-gr.bells;" described in §4.12 of the reference manual. Here, computation will not stop unless the buffer size limit is reached, or the mouse is clicked.

Incidentally, project ".i.-gr.cloches1;" uses a .i.homomorphism; labelled "TR" (see the alphabet window) similar to a .i.transposition;, although more complex. This may give you ideas for homomorphisms generating tonal transformations in a more complex way than the ".i.\_transpose;()" performance control. (Keep in mind that the same note may be played by different sound-objects.)

**4.8 The .i."Tuning" dialog**

Fig.23 displays a dialog named "Tuning" accessible from the "Misc" menu. By default, the middle C note ("C4" in English, "do3" in French, "sa4" in Indian notation) is assigned key 60, and the "A4" frequency is 440Hz.

See §1.11 the convention on octave numbering.

There was a confusion in previous versions of BP2 (until 2.6.2). Middle C had been assigned to C5 because of an error inherited from another music software, which went unnoticed until the Csound interface was checked. Therefore old projects may play one octave too high. To compensate this, assign middle C to key 48, and set A4 to 220Hz if you are using Csound. Change values in the "Tuning" dialog and save the project settings.

In a forthcoming version a transposition tool will be available to modify otave numbers in grammar, data and alphabet windows.

The .i.**diapason frequency**; of BP2 is used only for producing Csound scores with instruments that take the ".i.cps (Hz);" format as a pitch parameter.

**5. Non-stop improvisation**

**5.1 "Random" isn't anything...**

When producing items, BP2 made .i.random decisions; as to candidate rules in the grammar and the position of each derivation. These decisions may also be controlled in "step computation" (see §12).

Sequences of .i.random numbers; are predetermined in computers. This means that each time you run BP2 you get the same sequence. If you run an improvisation immediately after starting BP2 you get the same "random" performance... Of course, if the .i."Autorandomize" button; is checked on the upper .i."Settings" dialog;, every piece is likely to differ from tyhe preceding one, but this may not still not be the way you expect a machine to "improvise"!

There are two ways of avoiding this. First, you may want to instruct BP2 to produce anything you've never heard before. For this the .i.random-sequence generator; should be reseeded with an arbitrary number, for instance a number computed from the date/time information available inside the computer. Select ".i.Randomize;" in the "Misc" menu. You get the little "Randomize" dialog as shown Fig.23. The current .i.random seed number; is always 1 when you start BP2, therefore default random sequences are identical. The random seed is saved in the "-se" setting file automatically loaded with each project.

Clicking "Reset" will restart the random generator with the current seed number, thus producing again the same sequence of random numbers. Clicking ".i.New seed;" will take an arbitrary number as a new seed and recalculate the sequence accordingly. Now you get something really unpredictable.

Another way of playing with randomness is to type a seed and click "Reset". In this way you select manually one among more than 65,000 sequences (because the seed may be any number between 0 and 65535). If you get a remarkable result with a given value of the seed number, you should note it down carefully or save it along with the current settings (see the "File" menu).

There is no certitude that every future version of Macintosh® will produce the same random sequence starting from a given seed number. In addition, you must keep in mind that if you make changes in the grammar (modifying or moving rules) the sequence is likely to change. If you need more control on productions, either modify the grammar (see for instance "Programmed grammars", §11 infra, or control procedures, reference manual §8.1) or make decisions by hand as shown §12 infra.

**5.2 A kind of cellular automaton playing a kind of Turkish music**

Load ".i.-gr.koto3;" and type cmd-option space. Check "Non-stop improvize", "Use each substitution", and uncheck "Display items":



**Fig.24 Controlling production -- The** .i.**"Settings" dialog**;**s**

Now click "Produce items..." or type cmd-r. Since you selected ".i.Use each substitution;" you will see and hear all intermediate work strings produced by subgrammar #2. (See "SUB" grammars, §4.14 of the reference manual.)

All variables (like "X" and "Y") in the workstring are ignored when interpreting it as a musical item. Therefore, only terminal symbols such as 'a', 'b', 'c', 'chik'... will produce sound-objects.

While BP2 is improvising you may hold down the mouse-button to interrupt the process. This makes it possible, for instance, to uncheck "Show graphics" and check "Display items". Now all items played by BP2 are also written to the "Data" window.

Buttons appearing at the bottom of Fig.24 also belong to the "Settings" dialog:

• **".i.Use MIDI in/out;"** allows BP2 to send data to MIDI devices (including QuickTime music, if you are using Opcode OMS) and to receive messages from external MIDI devices (including the Inter-Application Communication bus, if OMS is installed). If this option is not set, then at least either ".i.Write Csound scores;" or ".i.Write MIDI files;" should be checked.

• **".i.Synchronize start;"** keeps BP2 waiting for either a NoteOn message, a .i.**"Start"** MIDI message; or .i.**"Continue"** MIDI message;, or a mouse click, before playing any item. It will wait for a NoteOn if and only if the key and channel are defined in the "-in.abc1" interaction file (see §6).

• **".i.Compute while playing;"** tells BP2 that it may go on computing items while the current one is being played. This is a good way to ensure continuous output, but remember that interactive commands do not affect items already queuing in the MIDI buffer. For instance, a "repeat" instruction will often not repeat the item currently being heard, but the one being computes.

 If this option is off in continuous improvisation, BP2 keeps a minimum number of items in the queue. For this, it remembers the maximum computation time of items already produced. Let *tmax* be that time. While the item is being played, BP2 will wait until the duration of the part still remaining to be played is a little more than *tmax*. Then it proceeds with a new item, thus hoping that it will just be ready to play it when the preceding item has been played entirely (see "-gr.Mozart" §5.4).

• **"**.i.**Interactive**;**"** sets the .i.interactive mode; on. The "-in.abc1" file is loaded and the MIDI input is used for controlling BP2 (see §6).

• **".i.Reset rule weights;"** is meaningful only if there are .i.dynamic weight; assignments in grammars, e.g. a weight written <100-20> in a rule that takes decreasing values 100, 80, 60,... every time the rule is used. (See reference manual §4.7) If it is on (default setting) then weights are set to their initial values each time a new item is computed from the start string.

• **".i.Reset flags;"** is meaningful in the context of .i.programmed grammar;subgrammars (see §11). Normally it is necessary to delete all existing flags before generating a new item, but the opposite option may sometimes be useful.

• When **".i.Ignore constraints;"** is on, the .i.time setting algorithm; does not take topological properties of sound-objects into consideration (see Bel 1990b,1991,1992, or reference manual §3-5).

• **".i.Write MIDI files;"** gives the option to save items as .i.MIDI file;s after playing them (see §14.1).

• **".i.Write Csound scores;"** gives the option to save items as .i.Csound score;s after playing them (see §17).

Other buttons shown on top of Fig.24 have the following effects:

• **".i.Cyclic play;"** produces a single item and plays it forever.

• **".i.Use each substitution;"**: this flag is used in "SUB" subgrammars (see §4.14 of the reference manual). If it is on, the current work string is played on the MIDI output after each substitution. Polymetric structures are evaluated and variables are ignored. If it off then only final items are played.

• **".i.Time setting display;"**, **".i.Time setting trace;"** and **".i.Time setting step;"** are optional controls of the time setting algorithm. These have been used, for instance, in example ".i.-da.checkTimeset;" (see §2.2 and §13);

• **".i.Trace Csound;"**, when checked, displays the Csound score on the "Trace" window while it is being recorded to a file.

**5.3 The "INIT" instruction**

In case you wish to play ".i.-gr.koto3;" on a Roland D-50 or D-550 synthesizer, you might insert the following line top of the grammar:

INIT: Play \_script(MIDI set basic channel to 15) Koto \_script(MIDI Omni mode ON channel 15)

What does it mean? 'INIT:' indicates something that is done during the initialisation, that is, every time the grammar is requested to improvise. Here, the initialisation is a script instruction "Play «any item»" in which the item contains two .i.**performance control;s** "\_script()" and a .i.**variable**; "Koto" (starting with an uppercase character).

Instruction "MIDI set basic channel to 15" tells BP2 that MIDI program changes should be sent on channel 15. These program changes are performed by the glossary when rewriting variable "Koto". The effect depends on the glossary which contains the proper program number setting a patch named "Koto". Then, "MIDI Omni mode ON..." will set the synthesiser to OMNI OFF, so that messages may be received and played even if they are not sent on channel 15.

.i.Initialisation; is performed before playing the first item produced by the grammar. If it contains a "Play" instruction, BP2 first produces the item, performs the initialisation, and then immediately plays the item. Otherwise it performs the initialisation before producing the item.

An example will make it clear. If the initialisation is

INIT: Wait for space

then BP2 will prompt you for hitting the space bar just after compiling the grammar, before producing an item. If it is

INIT: Play \_script(Wait for space)

then it will prompt you just before playing the first item the grammar has produced. This is a good way of synchronising the beginning of an improvisation. You may even want to be notified by a beep:

INIT: Play \_script(Beep) \_script(Wait for space)

The specificity of the "INIT" instruction should be made clear. You could as well insert the preceding instructions as .i.**performance control;s** into the item produced. For instance, the first rule in the ".i.-gr.koto3;" grammar could be:

S --> \_script(Beep) \_script(Wait for space) \_vel(127) \_volume(40) X X X X Y X X X X Y X X X X

However, in this case, the beep and wait would be performed each time a new item is produced, which is not convenient in the "Non-stop improvize" mode.

You may also use the "INIT" instruction to execute any script stored on the disk, e.g.:

INIT: Run script "MyInitScript"

**5.4 Mozart's musical dice game**

.i.musical dice game;You may not like the so-called Turkish music improvised on a so-called Koto... I would recommend listening to the marvellous CD "Masters of Turkish Music" (Rounder CD 1051) and apologise to Turkish music lovers and lovely Koto players for such a joke.

How about some *unknown* pieces (presumably) composed by Wolfgang Amadeus Mozart? Select "Load project" in the "File" menu (cmd-l) and load
".i.-gr.Mozart;".

This grammar uses "simple notes", i.e. the predefined alphabet of French notes. This convention has been automatically selected when settings "-se.Mozart" were loaded.

Now produce items (cmd-r)...

In the current set-up (mm = 220 ticks/min) the duration of any variation is about 26 seconds, slightly more than computation time on a Mac II-ci. Therefore, Mac II-ci and faster machines are able to string variations without break. You may also save time by unchecking "Show messages" in the "Settings" dialog.

.i.weights in grammar;Weights of some rules in the grammar are controlled by <K1>, <K2>, etc., and may be changed by MIDI controllers (see §6.5). Since controllers are not used here, these parameters are assigned the constant values defined in the first grammar. These actually reflect the probabilities of getting 2, 3, 4,..., 12, out of a two-dice throw. An explanation of weights may be found in the reference manual, §4.6.

The last subgrammar has instruction ".i.SUB1;", a faster version of ".i.SUB;" when substitutions are to be performed only once.

You may select .i.polymetric expression;s in the last subgrammar and play them (cmd-p) in order to listen to the various units defined by the composer. If the selection contains a variable it is used as a start string by the grammar. Try for instance to play "A1 A2 A3".

To select an expression with balanced curled brackets {}, click anywhere inside the expression and type cmd-b ('balance'). Repeat typing to extend the selection.

How was this grammar created? Subgrammars 1 to 17 have been directly typed from Mozart's tables. Subgrammar 18 contains the "terminals" of the

"dice grammar": musical segments taken from Mozart's score in conventional music notation. Selecting ".i.Type from MIDI;" in the "Action" menu, or typing cmd-j allowed BP2 to capture notes from a MIDI keyboard, and prolongations '\_' with the "hold" pedal.

If you are not yet familiar with polymetric structures, this grammar can teach you the essential: select structures, play them and compare .i.polymetric expression;s with their "scores" appearing on the "Graphic" window.

**6. Interactive control**

When the .i."Interactive (use input)" button; is checked on the lower .i."Settings" dialog; (accessed by typing cmd-option space, see Fig.24 in §5.2), BP2 keeps listening to messages received on its MIDI input and may react to them if instructed to do so. It keeps listening even whe it is not doing anything particular such as improvizing music or running a script.

Instructions on how to react are stored in a ".i.-in.<name>;" file displayed in the .i."Interaction" window;. The format of these instructions is part of BP2's script language, therefore it is convenient to say that the "Interaction" window contains a script.

Conversely, BP2 produces MIDI messages, some of which may be used to trigger processes on external devices (including copies of BP2). For instance, BP2 is able to start *any* process when prompted by the appropriate MIDI message. This is possible thanks to the instruction

IN On «note» channel «1..16» do «script instruction»

which will execute the specified script instruction at the time a NoteOn message is received, including running another script if «script instruction» is:

Run script «filename»

**Caution:** this instruction will only work if BP2 finds the specified script (or its alias), either in the start-up directory (the one in which you did a double-click to start BP2), or in the latest directory in which a script was found during the current session. Otherwise, BP2 will prompt the user to locate the script.

**6.1 When does it react?**

BP2 does react to MIDI input provided that (1) it is in the appropriate format, i.e. the message has been declared in the "Interaction" window ("-in.<filename>" file), and (2) both the ".i.Interactive (use input);" and ".i.Use MIDI in/out;" buttons on the lower .i."Settings" dialog; are checked (see Fig.24).

The status of these buttons is saved and loaded along with ".i.-se.<filename>;" settings files. Each time a project, grammar or data is loaded, their status might therefore change. Also make sure that a file has been loaded to the "Interaction" window, or load it manually if necessary: its name will be inserted on top of the "Grammar" and "Data" windows, so that it may automatically be loaded next time you start the project.

A third condition for BP2 to react to MIDI input messages is that it has free time. Messages are stacked into the MIDI input buffer waiting for BP2 to read them, which it does as often as possible. A delay in the response may be caused by the fact that BP2 was engaged on some more urgent task.

Apart from launching scripts, many parameters controlling BP2's operation may be directly modified via the MIDI input, as demonstrated below.

**6.2 The MIDI filter**

Let us check the ability of BP2 to receive and transmit MIDI messages. Select "MIDI filter" in the "Misc." menu.



**Fig.25 The MIDI filter dialog**

Checked boxes on the left indicate which types of MIDI messages are currently received by BP2. Some of these messages may activate processes, as we will see later. The right column indicates which messages are transmitted to the MIDI output. Transmission may be necessary if BP2 is part of a chain of MIDI devices -- unless you connect other devices to the "MIDI thru" output of the computer's interface.

If a box is checked in the right column the corresponding box is automatically checked in the left column, because a message cannot be transmitted unless it is received.

Settings shown Fig.25 are the default settings recorded in "-se.startup". If you load project ".i.-gr.koto3;" the MIDI filter dialog will display the settings found in "-se.koto3". Clicking the "Reset" button makes it possible to retrieve start-up settings.

BP2 keeps receiving and transmitting messages (at idle time) provided that the "Use MIDI in/out" button is checked on the top "Settings" dialog (type cmd-option space, see Fig.24). If the option is not checked you will be prompted to turn it on when displaying the MIDI filter dialog.

**6.3 The "Interaction" window**

Suppose that the project ".i.-gr.koto3;" has been loaded. Before playing the first item, BP2 loads interactive code file "-in.abc1" that contains the actual information the project needs for external control. The file is loaded just before the first item is played if "Interactive (use input)" is checked on the "Settings" dialog.

File "-in.abc1" appears as follows:

BP2 script

Note convention = English

IN Synchronize start ON-OFF F#4 channel 15 [toggle]

IN Repeat 'v' times D5 channel 15

IN End repeat G5 channel 15

IN Repeat forever E5 channel 15

IN Derive further A4 channel 15

IN Reset weights B4 channel 15

IN Quit F5 channel 15

IN Use each substitution G#4 channel 15

IN Set computation time to 'v' F4 channel 15

IN Smooth-striated time E4 channel 15

IN Use-ignore object constraints C#5 channel 15

IN Skip next item D4 channel 15

IN Play again item D#5 channel 15

IN Min 10 ticks in 3 sec C6 max 20 ticks in 3 sec C7 channel 15

IN Synchro tag W1 = C3 channel 15

IN Synchro tag W2 = C#3 channel 15

IN Synchro tag W7 = D3 channel 15

IN Synchro tag W8 = D#3 channel 15

IN Parameter K10 = velocity C8 channel 15

IN Param K1 = controller #7 channel 15

IN Param K2 = controller #17 channel 15

IN Param K9 = controller #0 channel 15

This file is actually a **script**. Instructions appearing in the "Interaction" window belong to the subset of well-formed script instructions. (See the entire set in the .i."Script" menu;.)

The order of instructions in the "Interaction" window is not relevant. The only instructions that should appear on top are "BP2 script" and "Note convention = ..." indicating the convention used for interpreting the following instructions. If another note convention is selected in the "Misc" menu, the content of the "Interaction" window is automatically translated to the new convention, and the second instruction is modified or created accordingly. In this way, interactive files remain consistent with projects using different note conventions.

The compiler checks the consistency of instructions: if for instance the same key or controller is assigned several tasks, an error message will be displayed.

New instructions are entered by pulling down the "Script" menu when the "Interaction" window is active. Slots may then be filled with the required values. Remember that you can enter note names by hitting a key on the MIDI keyboard once ".i.Type from MIDI;" (cmd-j) is active. All instructions are documented on-line: type cmd-? and pick an instruction in the "Script" menu.

**6.4 Distributed improvisation**

Interactive controls make it possible for several BP2's and other MIDI devices to collaborate on the same MIDI 'network'. Communication is generally done on a reserved MIDI channel, here for instance channel 15. **You must set the MIDI keyboard to channel 15** to be able to interact with project ".i.-gr.koto3;".

Consider for example:

IN Synchronize start ON-OFF F#4 channel 15 [toggle]

IN Start play C5 channel 15

The first instruction (a toggle command) assigns key F#4 to instruct BP2 to accept or ignore 'Start play' instructions. When BP2 starts improvising, its response to 'Start play' depends on the setting of radio button "Synchronize start" on the "Settings" dialog. (It might also have been changed by script instructions "Synchronize start ON" or "Synchronize start OFF".) It will be changed again by key F#4 channel 15.

The second instruction is the 'Start play' definition itself: BP2 should wait for a NoteOn of key C5 channel 15. If this instruction is missing, then instead of waiting for a NoteOn BP2 will wait for a 'Start' or 'Continue' MIDI message.

Note that this amounts to the same as inserting a performance control "\_script(Wait for C5 channel 15)" or "\_script(Wait for Start)" in the beginning of each item. However, once defined in the "Interaction" window, the 'Start play' instruction may be deactivated and reactivated by key F#4.

'Start play' is useful for synchronising the beginning of a musical item accurately. The 'start' message may be sent from a MIDI keyboard or from any other MIDI device, including another BP2. For instance it could be (it *is*!) the 'sync' sound-object terminating every production in grammar ".i.-gr.koto3;". With two computers connected to MIDI, you may run project "-gr.koto3" on both and see how they collaborate... Don't forget to set up "Interactive (use input)" on both machines. Then modify the grammars to give an 'individuality' to each one. It works? You've entered the world of *distributed* *improvisation*! Can you figure out a set-up with more computers, including humans messing around?

Sending a NoteOn on key C5 channel 15 otherwise prompts BP2 to play an item immediately. If BP2 is still busy computing the item, several flags will be set so that production is terminated as quickly as possible (by-passing all constraints). This is an equivalent of "it's up to you now" signals used by jazz musicians in a band.

**6.5 More interaction!**

Here are a few hints for using the interactive features of BP2 in a set-up that would comprise several computers with various MIDI software, and human musicians interacting with keyboards or MIDI instruments.

Limitations are mainly with MIDI channels: several channels are required to play different instruments or to connect different keyboards, and a few remaining channels may be assigned to "non-musical" communication. Don't forget to set MIDI devices to .i.OMNI OFF; so that control messages are not audible.

Let us look at a few instructions contained in "-in.abc1".

IN Set computation time to 'v' F4 channel 15

instructs BP2 that every NoteOn received on F4 channel 15 assigns a limit value to the .i.**computation time**; of the current item. The limit is a function of the NoteOn's velocity. This is a way to tell BP2 it should come up with an item ready to play before some other process has been completed. (For instance another BP2 is currently playing the preceding item and it is able to figure out its maximum duration.) If *v* is the velocity of the NoteOn on key F4, computation time may not exceed 472 *v* (in milliseconds).

IN Min 10 ticks in 3 sec C6 max 20 ticks in 3 sec C7 channel 15

instructs BP2 that all keys between C6 and C7 on channel 15 contain a new setting of its time base (and metronome). Here, C6 will set the time base to 10 ticks in 3 seconds, and C7 to 20 ticks in 3 seconds. Keys between C6 and C7 will assign interpolated values.

A 'change tempo' instruction could be imbedded in a particular musical item. Thus, BP2 may instruct its neighbours that the metronome has changed. It may also change its own metronome using "\_script(Tempo ... ticks in ... secs)". A good idea is to follow the 'change tempo' instruction with a 'skip next item' instruction using key D4 channel 15 as instructed by

IN Skip next item D4 channel 15

IN Set tempo controller 0 channel 15 range 4.5

The ".i.Set tempo;" controller is a MIDI pedal changing the metronome speed. "Range 4.5" is the ratio between the maximum value and the minimum value. If the controller is in its medium position (64) then the metronome value is the default one.

IN Parameter K10 = velocity C8 channel 15

IN Param K1 = controller #7 channel 15

These instructions allow the setting of .i.**control parameter;s** K10 and K1, respectively, by MIDI devices: K10 is set to the velocity of a NoteOn message on key C8, channel 15, and K1 is set to the value of MIDI controller #7, channel 15.

Remember that K1,... are global variables that may be used as .i.**weights in grammar**; rules, arguments of .i.**performance control;s** (e.g. \_volume(K10), see §1.10), or comparison values in flags (see §11.2).

**6.6 Synchronisation tags**

Consider the following instructions:

IN Synchro tag W1 = C3 channel 15

IN Synchro tag W2 = C#3 channel 15

IN Synchro tag W7 = D3 channel 15

IN Synchro tag W8 = D#3 channel 15

These define .i.**synchronisation tag;s** notated <<W1>>, <<W2>>... Each tag is assigned an .i.**input sound-object**;.

When BP2 finds a tag within a structure of sound-objects it stops everything (including its internal clock) until the input sound-object is received, i.e. the corresponding NoteOn is received or the mouse is clicked. This allows you to synchronise any performance on notes or sequences of notes expected at a particular time.

Suppose that you have typed or produced the following item:

a b <<W7>> c <<W1>> <<W3>> <<W8>> d

(Use example in file ".i.-da.trytags;") BP2 will play 'a' and 'b' normally, then it will display a message indicating that it is waiting for "D3 channel 15", i.e. the <<W7>> tag. You may actually depress the "D3" key , send a NoteOn or simply click the mouse. BP2 will then continue playing 'c' and hang again waiting for C3, the <<W1>> tag. Now it must stand on the <<W3>> tag, which is not defined, so it prompts you to send a mouse click. Eventually, it will hang on <<W8>>, prompting you to play D#3 or send a mouse click. When done, BP2 will play 'd'.

.i.Synchronisation tag;s may appear in .i.polymetric expression;s as well. Each tag relates to the sound object immediately following it in the same sequence. For instance, the item

b b <<f>> {5, a c b, f - <<W8>> f} a <<chik>> b <<sync>>

yields exactly the graphic output shown Fig.17, but the <<W8>> synchronisation tag will force BP2 to wait for a click or D3 channel 15 before it performs the second occurrence of 'f'. The score table below (Fig.26) makes it clear that the actual interruption separates the item in two parts:



**Fig.26 A phase diagram for
"b b <<f>> {5, a c b, f - <<W8>> f} a <<chik>> b <<sync>>"**

Synchronisation tags are processed as .i.**out-time sound-object**;**s** but they represent "input" instead of "output" sound-objects. They are also less complex than other out-time objects because they may only be defined as NoteOn's. Undefined tags react to mouse clicks.

Synchronisation tags have been implemented in an earlier version of BP2, at a time scripts and performance controls were not available. A tag like <<W8>> could equivalently be replaced with any variable rewritten as:

\_script(Wait for D#3 channel 15)

**6.7 Apple Events -- BP2 in a client-server environment**

Bol Processor BP2 is designed for .i.**client-server technology**;.

First of all, BP2 can run in background. For instance, after launching an improvisation or a script, clicking outside BP2 (in the Finder or any other application window visible in the back) sends BP2 to **background**. This means, it continues its job, including playing sounds through the MIDI output, while the user is doing (almost) anything else on the same computer. Indeed, running several applications implies that enough RAM space is available and the machine is good at sharing resources in an efficient way.

.i.Apple Event (receiving);In addition, BP2 responds to .i.**Apple Events**;, i.e. events that enable communication between different applications under MacOS. A typical "client" application is supplied to show the range of possibilities. It is a Claris® FileMaker Pro database named ".i.TryAppleEvents.fm;", which contains scripts telling BP2 to do certain things, and explanations on the actual operation.

If it does not work, check Apple Events created by the FileMaker scripts (call "ScriptMaker" in the "Script" menu of FileMaker 3.0). Each event appearing in the script should show "BP2.9.3" as the target, not "<unknown>". If it shows "<unknown>" it probably means that ".i.TryAppleEvents.fm;" is not in the same folder as "BP2.9.3", or the desktop needs to be rebuilt.

Note that FileMaker Pro 3.0 does not recognise an application on a remote disk, which means it is not able to send a remote Apple Event even though BP2 would be able to process it.

The environment should be arranged properly when BP2 is used as a server. It is recommended that no client application hides the bottom part of the screen on which BP2 displays messages. The top left part is also useful to see that BP2 has received a "pause" event and is waiting for "resume" or "stop". Doing so, its activity can be controlled without bringing BP2 to the foreground.

The list of currently available Apple Events is found in the on-line documentation under the "Apple Events" entry accessed in the "Frequently aked questions..." dialog. The current version processes events sending musical 'scores' in text format, BP2 scripts, and various tasks such as loading a settings file or changing the note convention.

The .i.client-server technology; has tremendous potential. BP2 is so far a unique MIDI software able to handle musical material as **text** (scores and

*processes* such as grammars or scripts). This data may be stored in the fields of a standard database working as a client of BP2.

.i.Apple Event (sending);BP2 may in turn **send .i.Apple Events**; to other applications (running on the same computer) thanks to script instructions:

AE send normal class '«AEclass»' ID '«AEID»' to application '«signature»'

AE send fast class '«AEclass»' ID '«AEID»' to application '«signature»'

The difference between these two instructions lies in the priority which the Finder will give to the event.

Note that in the current version, the target application must already be opened, failing to which the Apple Event won't be able to reach it; in other words, BP2 does not launch applications.

If you are working in the field of graphic animations you may take advantage of BP2 as a server for producing high-quality music on MIDI devices. Random improvisation may be an important asset: each time the animation is played part of the music track will be composed afresh.

**7. Glossaries**

Two ready-made glossaries are supplied with BP2: ".i.-gl.D50;" and "-gl.GeneralMIDI". The first one contains information relative to Roland D-50 synthesiser, and the second one information that is common to all .i.**General MIDI**; synthesisers. If your synth is neither Roland D-50 nor General MIDI, you may edit a specific glossary on the basis of supplied examples.

A .i.**glossary**; is altogether a grammar and a script... Let us first look at it as a grammar. Load ".i.-gl.GeneralMIDI;" to the glossary window for instance:

// This glossary defines program changes used to select instruments following General MIDI specifications.

// Note that MIDI programs are numbered 0 to 127 here (following MIDI spec. 1.0) instead of 1 to 128 as per General MIDI specs.

// You may change variable names, but names should remain consistent in all your projects. (See acceptable variable names under "variable" in the Help menu.)

// Beware that program change messages will be sent on the current channel, which should be the basic channel of the synthesizer (i.e. the one on which it receives mode messages). This channel (default 1) may be changed by script instruction: MIDI set basic channel to ...

[1] AcousticGrandPiano --> \_script(MIDI program 1)

[2] BrightAcousticPiano --> \_script(MIDI program 2)

[3] ElectricGrandPiano --> \_script(MIDI program 3)

[4] HonkyTonkPiano --> \_script(MIDI program 4)

[5] RhodesPiano --> \_script(MIDI program 5)

[6] ChorusedPiano --> \_script(MIDI program 6)

[7] Harpsichord --> \_script(MIDI program 7)

...

It clearly looks like a grammar (a subgrammar) with its rewrite rules. Because of speed optimisation it is always taken as a 'SUB1' type subgrammar, i.e. it performs one single substitution of the start string. Rule syntax is limited: the left argument should contain a single variable. If that variable already appears in the .i.left argument of a rule; belonging to the grammar, the compiler warns you that it may be a mistake, but proceeds anyway. If the same variable is defined twice in the same glossary, or if the glossary generates undefined variables, then en error message is returned.

How does it work practically?

Suppose that BP2 has produced an item and is about to play it, or you have selected an item in the text window. It first scans the item looking for variables that are used in the glossary. If no such variable is found then the glossary is not used. Otherwise it uses the glossary as a subgrammar in order to get the final string, which will then be expanded (as a .i.polymetric expression;), set in time, and then played on the MIDI output.

Since the glossary makes a single substitution, a variable appearing in the .i.right argument of a glossary rule; will only be rewritten if it is defined in a following rule. (See for example variables 'Internal' and 'Expansion' in "-gl.D50".)

Let us now look at the .i.**glossary as a script**;. Any rewrite rule may equivalently be put as a "Define" script instruction, e.g.

ChorusedPiano --> \_script(MIDI program 6)

is equivalent to:

Define ChorusedPiano \_script(MIDI program 6)

In addition, script instructions that influence compilation are acceptable in glossaries, notably:

Note convention = key numbers

You may use glossaries for any other purpose than setting a MIDI device. The good thing about glossaries is that they can perform transformations on a string representing a musical item even when no grammar is used to produce it. It makes sense that the design of device-specific procedures is not part of a grammar which is only meant to produce symbolic representations of musical items.

**8. Scripts**

Scripts have been briefly introduced in §1.3. A script is a sequence of instructions that may be executed to replace manual operations on BP2. AppleScript and HyperTalk are good examples of script languages used by the Macintosh®. Most recent software environments are scriptable, some even are accessible from AppleScript.

The only link between BP2 scripts and the Mac operating system is through BP2 script files (or their aliases). In other words, if AppleScript sends a double-click to a BP2 script, then BP2 will automatically be launched and the script will be executed. However, AppleScript will loose the control until the script has been entirely executed and the "Quit" instruction has been found. The script could for instance load some data and/or grammars, produce musical items, print windows, etc., and then return to AppleScript. See for instance ".i.+sc.checkAll;" which is used to verify most functions of BP2.

A later version of BP2 might be "scriptable", i.e. operated step by step from AppleScript.

(Prefix "+sc" was chosen instead of "-sc" to avoid confusion with "-se".)

Unlike AppleScript and HyperTalk, the current version of BP2 script language does not accept variables and has no conditional branching ('if', 'repeat'...).

BP2 executes scripts in two ways: either it reads them in the "Script" window, or it reads them directly from disk. In the first case, the current instruction is hilited in the "Script" window, in the second case it appears in the "Message" window (the bottom line of the screen). Execution may also be done step by step.

Script instructions are listed in the "Script" menu.

Although the script language comprises already 161 instructions, it is still under development. Priority has been given to automating music performance, rather than edit procedures.

**8.1 Checking a script**

Before executing a script for the first time it is recommended to select "Check current script" in the "Script" window. This debugging tool does not only checks script syntax. It also checks that all files loaded by the script are accessible. If another script is called by the current script, it is loaded and in turn checked. The debugger also makes sure that the same script will not be called twice during the current script's execution (thus avoiding recursive calls). If a file is not found, the script debugger prompts the user to locate it. Then it may rename the file or keep trace of its location by inserting instructions like "Set directory..." or "Set Vref...". Understandably, it is necessary to call "Check current script" every time once of the files used by the script has been renamed or relocated.

**8.2 Scripts, volumes and folders**

Although BP2 scripts can load files from different folders, and even from remote disks, it is recommendable to keep all concerned files in single project folder. An excellent method is to keep files where they are and to place their aliases into the project folder, as shown Fig.10.

At execution time, BP2 does its best to identify folder(s) in which the files required by the script will be found. For example, if a grammar file has been loaded from a specific folder, then other grammar files will be searched in that same folder. In addition, the folder in which the current script is saved will be searched. Instruction "Set directory..." will provide the path to a specific folder.

Instruction "Set Vref..." is less practical because the value of Vref depends on the volumes currently opened. If a script contains "Set Vref..." it may not execute properly if you start your computer in a different configuration, e.g. the Syquest unit or the scanner is not switched on. If the script debugger generates a "Set Vref..." instruction it is therefore recommendable to delete it and locate the file (or an alias of it) on the same volume as the script.

**9. Make it swing with smooth time...**

So far we played around with musical items that were set on .i.**striated time**;, i.e. with a regular beat imposed by the time base (metronome, see §4.3). Indeed it

was possible to change the metronome setting (although not in the middle of an item) and the time setting algorithm provided for infinite flexibility with sound-objects relocated or truncated on the basis of individual properties. However it seemed impossible to program an *accelerando* or some non-metric time-pattern like the typical elongated third beat in Saturday evening's waltz.

With .i.**smooth time**; you will be able to combine "plasticity" with accuracy. Suppose for instance that you want BP2 to play

do5 re5 mi5 fa5 - la5 si5 do6\_ mi6

(in which '-' is a silence and '\_' is the prolongation of do6) with the following time-pattern:



**Fig.27 Time-pattern with** .i.**irregular beats**;

"t1", "t2", etc. are .i.**time-object**;**s**, somehow similar to .i.**sound-object**;**s** except that they do not contain any sound-generating code. Here the piece is divided in two sections, namely "t1" and "t2", each of them lasting five beats with different **physical durations**.i.physical duration of sound object;. Let us assume that "t1" is a "normal duration" yielding five beats in 2.5 seconds, therefore we will set mm = 120. Here the metronome does not construct a regular pattern of .i.time streak;s. It only serves as a time reference. This case is referred as .i.**measured smooth time**;. The second section "t2" is also five beats but it lasts 3.75 seconds, i.e. 1.5 times the first section. When specifying time-patterns each duration will be written as an integer ratio. We will therefore write:

t1 = 1/1 t2 = 3/2

(We could write t2 = 150/100 or any equivalent ratio as well.) This statement contains three pieces of information: (1) t2 is 1.5 times longer than t1, (2) the default duration of t1 is one metronome beat, (3) the default duration of t2 is 3/2 metronome beats. Of course, (3) may be deducted from (1) and (2).

Let us now look at subdivisions. "t1" contains three subdivisions labelled "t1", "t3" and "t4". Of course, the new "t1" has been resized so that the three subdivisions fit exactly in the first section. Only ratios are important at this stage. You may measure that t3 is 1.33 times longer than t1, and t4 is two times shorter. Therefore we will write

t3 = 4/3 t4 = 1/2

(or equivalent integer ratios). The second section is subdivided as "t3" and "t1" in which the ratio between t3 and t1 is 1.33 as expected.

So far we have been able to define the .i.**hierarchy of time-span intervals**; as a .i.**tree structure**;. Let us now look at the note sequence itself. Time-object "t1" is subdivided as "do5" and "re5" with equal durations because all sound-objects belonging to "-mi.Frenchnotes" have identical time references -- corresponding to ratios 1/1. "t3" is subdivided in a similar way, as well as "t4" which contains '-' (a silence) and "la5". The last part "t3 t1" is an interesting case since the time-span of "do6" is partly a subdivision of "t3" and of "t1". This indicates that the hierarchy of time-span intervals is not always a tree structure -- here it is called a .i.**join-semilattice**;.

**9.1 Programming time-patterns**

A grammar producing the item shown Fig.27 may be written as follows (see ".i.-gr.tryTimePatterns;"):

PATTERNS:

t1= 1/1 t2 = 3/2 t3 = 4/3

t4 = 1/2

---------------------

RND

S --> {10,t1 t2,Part1 Part2}

Part1 --> {t1 t3 t4, do5 re5 mi5 fa5 - la5}

Part2 --> {t3 t1, si5 do6 \_ mi6}

Title "PATTERNS:" announces a list of .i.**time-pattern**; **definitions**. The list terminates with a string of '-' or the end of the grammar file. ".i.PATTERNS:;" definitions may also be inserted in the alphabet file. If so, the defined time-patterns will be used with all grammars based on the same alphabet. Patterns defined in the grammar are specific to productions of that grammar.

If you load only data (e.g. ".i.-da.tryTimePatterns;") and try to play a selected item, BP2 will first look for time-patterns in the alphabet, then in the grammar file. If ".i.-gr.tryTimePatterns;" has not been loaded, the compiler will complain that *terminal* symbols "t1", "t2", etc. are undefined.

You may use any label for time-patterns, including labels starting with an uppercase character, but then the compiler will not return an error message in case these patterns are undefined, as it will mistakenly take them as variables.

It is very important that .i.time-patterns in polymetric expression; appear in the leftmost position of .i.polymetric expression;s (see detailed explanation §4.10 of reference manual). Here, "t1 t3 t4" and "t3 t1" are the first fields of .i.polymetric expression;s, and "t1 t2" is the first field because duration "10" should always be the first field. Suppose for instance that you change the second rule to:

Part1 --> {do5 re5 mi5 fa5 - la5, t1 t3 t4}

This would yield:



**Fig.28 An incorrect use of time-patterns t1, t3, t4**

in which duration ratios in section "t1 t3 t4" have been mistakenly determined by "do5", "re5",... "la5". In addition, .i.symbolic duration;s are incorrect because Part1 now counts six units (do5, re5,...) instead of three (t1, t3, t4). Now the .i.symbolic duration; of the entire item is now 6 + 2 = 8 instead of 3 + 2 = 5. BP2 therefore plays eight time units in ten beats, so that all on-settings of notes fall off-beat. For this reason most .i.time streak;s in Fig.28 are not numbered. This explanation requires a good understanding of polyrhythms in polymetric structures, but playing with the software will be certainly beneficial in acquiring it.

A (musically disastrous) example of time-patterns is found in ".i.-gr.MozartExpression;":

TIMEPATTERNS:

t1 = 96/100 t2 = 100/100 t3 = 102/100 t4 = 100/100

t5 = 100/100 t6 = 102/100 t7 = 104/100 t8 = 106/100 t9 = 107/100

t10 = 108/100 t11 = 107/100 t12 = 106/100 t13 = 104/100 t14 = 102/100

t15 = 1/1

-------------------

RND [Select rules randomly]

S --> {96,Tp1,A B}

A --> {12,(=Tp2),A1 A2 A3 A4} A5 A6 A7 A8 {12,(:Tp2),A1 A2 A3 A4} {12,Tp3,A5 A6 A7 A'8}

B --> B1 B2 B3 B4 B5 B6 B7 B8 B1 B2 B3 B4 {12,Tp2,B5 B6 B7 B8}

-------------------

RND [Here we define time-patterns]

<20> Tp1 --> Tp2

<30> Tp1 --> t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15

Tp2 --> t1 t2 t3 t4

Tp2 --> t1 t7

Tp3 --> t2 t1

Tp3 --> t14 t15

-------------------

etc...

We suppose that the entire piece is controlled by a general .i.time-pattern; "Tp1" that may be subdivided in either 11 or 4 sections. The value "11" is arbitrarily chosen here to demonstrate that time-pattern sections may not coincide with metric divisions (here the actual symbolic duration is 96 beats). Part A (48 beats) comprises four sub-sections. The first and third subsections are controlled by two identical occurrences of time-pattern "Tp2" that may in turn be expressed as "t1 t2 t3 t4" or "t1 t7". Occurrences are identical because of the use of Bol Processor typical .i.bracketing; "(=Tp2)...(:Tp2)". (See reference manual §4.3) This also shows that the same .i.time-pattern; "Tp2" may be used at different hierarchical levels. The fourth subsection of A is controlled by "Tp3", and the second subsection is not controlled, so it is played at a regular

tempo determined by whichever time-pattern has been used at the higher level. Section B comprises a first subsection lasting 36 beats at regular tempo, and then a second one lasting 12 beats controlled by "Tp2" (which may not be the same occurrence used in A).

Fig.29 shows the beginning of a variation in which overall time-pattern "Tp1" is subdivided in 11 sections "t5 t6..." and "Tp2" is rewritten as "t1 t2 t3 t4". Change of tempo is almost unnoticeable because time ratios are close to 1/1.


**Fig.29 Time-patterns in Mozart pieces**

**9.2 Loading time-patterns**

Time-patterns may be entered with a MIDI instrument. Bring to front either the "Grammar" or "Alphabet" window and select ".i.Load time-pattern;" in the "File" menu. The rest is self-explanatory: BP2 captures notes played on the MIDI input and stores their relative durations.

Note that it is necessary to play an extra note at the end because durations are measured on the basis of NoteOn messages. NoteOff's are ignored. It is therefore evident that *n* notes define *n-1* time intervals.

Each .i.time-pattern; may be assigned a variable. BP2 suggests labels "Tp1", "Tp2", etc. and inserts automatically a rule rewriting "Tp..." as the sequence of durations entered.

Notes used for defining time-patterns must be played in sequence: playing chords would mess durations because simultaneous events actually do not produce simultaneous MIDI codes. You may for instance play the Mozart piece "with one finger" (plus the extra note at the end) and store the 96 beats obtained as "Tp1". All variations will then be played exactly with the same expressive (or dull) duration ratios. You may also play parts of the piece several times and store them as optional time-patterns randomly selected by the grammar. With context-sensitive or programmed grammars (see §11) musically significant effects could be obtained.

**9.3 Non-measured smooth time**

When using .i.**non-measured smooth time**;, durations are determined exclusively by the durations of the prototypes of sound-objects found in the first sequence (appearing on the top of graphics).

To set this mode type cmd-t and type "0" in the "ticks" field of the .i."Time base" dialog;. The .i."Metronom" dialog; will display "no clock".

You may try to play Mozart pieces in .i.non-measured smooth time; if you want to make them sound as if your little sister were deciphering the score. This mode is useful when dealing with sound-objects that have predetermined durations.

**10. For BP1 old-timers...**

Users of BP1 (the Apple IIc version) will be happy to find all features of BP1 implemented in BP2 (version 2.4, November 1993).

**10.1 Tempo marking**

The first rules of ".i.-gr.dhin--;", a grammar producing a North Indian .i.*qa'ida*; (a set of variations used by tabla drum players) look like:

S --> 4+4+4+4/6 S192

S192 --> etc.

...

The expression "4+4+4+4/6" indicates how musical items derived from "S192" should be displayed. In earlier versions of BP2 (before BP2.5.2) it was notated "4/4/4/4/6"; although this notation is still valid it should be abandoned for the sake of clarity.

First consider "/6". This is an .i.**explicit tempo marker**;, indicating that each beat of the metronome contains six sound-objects. (This is referred as .i.*chegun* ;by Indian musicians.)

The sequence "4+4+4+4" indicates how beats should be grouped in a measure (*tala*) containing 16 beats (referred to as .i.*tintal*;). Here the measure is divided into four sections of four beats. Consequently, a musical item produced by this grammar may for instance be displayed:

dhin--dhagena.dha--dhagena.dhatigegenaka.dheenedheenagena.¬

tagetirakita.dhin--dhagena.dhatigegenaka.teeneteenakena.¬

teeneteenakena.dheenedheenagena.dheenedha-dheene.dheenedheenagena.¬

tagetirakita.dhin--dhagena.dhatigegenaka.teeneteenakena.¬

tin--takena.ta--takena.tatikekenaka.teeneteenakena.¬

taketirakita.tin--takena.tatikekenaka.teeneteenakena.¬

teeneteenakena.dheenedheenagena.dheenedha-dheene.dheenedheenagena.¬

tagetirakita.dhin--dhagena.dhatigegenaka.dheenedheenagena

Beats are separated by .i.**periods**; (that could be replaced with tabulations in a word processor), and sections of the measure are displayed on different lines. Here the item covers two measures, i.e. eight lines containing 32 beats and 32x6 = 192 sound-objects (as expected).

Note that periods before line breaks '¬' prevent the last beat from being merged with the first one of the next line.

If the *tala* is .i.*dhamar*; (14 beats divided 5+2+3+4) and tempo is .i.*tigun*;, the header will be "5+2+3+4/3".

If .i.periods; are inserted in data or grammar arguments, they are used by the compiler to change the tempo. For example, if a,b,c are terminal symbols,

abba.bcca.bcca.abc.ccb

will be interpreted as

{1,abba} {1,bcca}{1,bcca}{1,abc}{1,ccb}

and performed like:

/4 abbabccabcca /3 abcccb

The .i.polymetric expression; "{1,abba}" (meaning: "abba" is performed during 1 beat) is more flexible than "/4 abba". The former is a relative, and the latter an absolute, tempo specification.

Conversely, if a string of data is selected and ".i.**Show periods**;" is invoked (see the Control panel, type cmd-=), beat and section markers are automatically inserted. For instance,

 3+4+2/4 abbabccabcca /3 abcccbaab /1 bbb

will be rewritten as:

abba.bcca.bcca.¬

abc.ccb.aab.b.¬

b.b

(See more examples in ".i.-da.ShowPeriods;".)

**10.2 Parsing items**

Parsing ("Analyze selection" in the "Action" menu) is the attempt to check whether or not the arbitrary item you selected might have been produced by the grammar. Parsing is also called a .i.**membership test**;. There are severe restrictions in the grammar format that make this test valid. When compiling a grammar BP2 checks whether it is a "**true" BP grammar**.i.true BP grammar; (see reference manual §7) and then only it authorises parsing.

Grammar ".i.-gr.dhin--;" is an example of a "true" BP grammar. First you should compile the grammar (cmd-k) and generate its .i.**template**;**s** (see Control panel or "Action" menu). Then produce a few items in the "Data" window. You will notice that the produced items are displayed without any .i.structural marker; (brackets indicating repetitions, etc.). This display is a comprehensive "score" for Indian drum players. Structural markers will be automatically inserted when matching items against the .i.template;s.

Select one or several items and select "Analyse selection" in the "Action" menu (or click the same on the Control panel). BP2 will prompt you to use templates. Answer "yes". You may then decide whether you want to find only the first template matching each item and yielding a positive .i.

membership test;, or check all templates to find which ones are acceptable. While the first method yields a quick assessment of the item in reference to the language generated by the grammar, the second will also point out structural ambiguity, e.g. the same item may be assessed "correct" by the grammar while matching several templates:

>>> Analyzing item: /6 dhin--dhagenadha--dhagenadhatigegenakadheenedheenagenatagetirakitadhin--dhagenadhatigegenakateeneteenakenadhagenadhin--dhagenadha--dhagenadha--dhagenadha--tagetirakitadhin--dhagenadhatigegenakateeneteenakenatin--takenata--takenatatikekenakateeneteenakenataketirakitatin--takenatatikekenakateeneteenakenadhagenadhin--dhagenadha--dhagenadha--dhagenadha--tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Item matched template [15]

>>> Item matching template [15] accepted by grammar...

Item matched template [19]

>>> Item matching template [19] accepted by grammar...

Item matched template [20]

>>> Item matching template [20] accepted by grammar...

**10.3 Learning .i.weights in grammar;**

The process of .i.**weight inference**; has been introduced in (Kippen & Bel 1992). Given a set of items accepted by the current grammar it is possible to infer rule weights from the set of examples. The weight of each rule will be the number of times it has been used in parsing the sample set. Weight inference is used to improve the quality of randomly generated musical items by increasing the probability that these items resemble a set of items given as an input by an expert musician. See for instance the weights of rules in "-gr.dhin--".

Try this procedure as follows. Load ".i.-gr.dhin--;" and ".i.-da.dhin--;". Compile the grammar and produce its templates. (It's a good idea to save the current grammar with its templates, but remember that .i.template;s should be computed again each time structural rules have been modified in the grammar.) Now click "Save weights" on the Control panel to save current grammar weights in a "-wg" file. Then click "Set weights" and reset all weights to 0. Select all items in the "Data" window, click ".i.Learn weights;" and answer default options in all dialogs.

**10.4 Keyboard encoding**

It is possible to reprogram the keyboard in order to type certain strings ("tokens") with a single key stroke. This is generally used for the names of terminal sound-objects. Current keyboard encoding is visible in window "Keyboard" and may be saved with prefix "-kb". The name of the corresponding file (e.g., "-kb.kathak.azerty") is automatically inserted on top of the alphabet file. Conversely, inserting a name will prompt BP2 to load the declared "-kb." file as soon as the alphabet file is loaded..i.keyboard encoding

Using window "Keyboard" is self-explanatory. To activate/deactivate encoding you may switch button "Use these tokens" on and off, or check "Use tokens" / "Type text" in the "Misc" menu, or type cmd-option t.

Only 52 characters may be mapped to tokens. Any string shorter than 255 characters may be mapped to a key as a token. Fig.30 shows the encoding used for Kathak material on an English keyboard (QWERTY).



**Fig.30 Keyboard encoding dialog**

Keyboard encoding in versions ≥ 2.5 takes different keyboard layouts into account. Two layouts are currently supported: US (QWERTY) and French (AZERTY). The default layout is QWERTY, but if you change it to AZERTY when tokens have been defined, then the mapping is modified automatically so that tokens appear at the same locations on the keyboard -- except for the ones mapped to the 'M' key, which cannot be remapped. (When changing from AZERTY to QWERTY and vice-versa, the 'M' key is swapped with another key that does not contain alphabetic characters.)

The AZERTY/QWERTY option is saved along with the -kb.<filename> keyboard encoding.

**10.5 Documentation of Indian music**

An example of keyboard encoding is the material contained ".i.-da.kathak tihais;". When this data file is opened, it automatically calls the "-ho.kathak" alphabet which in turn opens the "-kb.kathak.qwerty" encoding. (You may change it to AZERTY if your keyboard is in French.) All tokens are *bol*s used by (Lucknow style) .i.Kathak dance;rs in North India, in a comprehensive transliteration designed by Andréine Bel. Numbers used as terminal symbols (in the so-called "number *tihais*" are notated #1, #2, #3... (pronounce *ek, do, tin*... in Hindi, otherwise *one, two, three*...) because ordinary integers are reserved for indicating durations.

Indian rhythmic material lends itself to descriptions by formal grammars, as exemplified by grammars ".i.-gr.dhati;", ".i.;-gr.dhin--", ".i.-gr.dhadhatite;" and

their variants able to produce .i.*qa'ida*;s (theme and variations pieces) in the .i.Lucknow style of tabla; drumming. Most grammars supplied as examples have been commented in our publications.

A grammar producing a famous Kathak .i.*number* *tihai*; composed by Pandit Birju Maharaj will be found in ".i.-gr.12345678;". More .i.*tihai*;s have been implemented in ".i.-gr.ShapesInRhythm;". If you need to play *lahra*s (cyclic melodic phrases) to practise Indian percussions, use ".i.-da.lahras;".

Indian raga music is also a good candidate for composition/improvisation models based on grammars and automata. The subtleties of *alankara* and *gamaka* may be rendered by performance parameters. Examples are still lacking since .i.performance control;s have been introduced recently. When Kumar S. Subramanian spent a few days in Delhi, he started composing items related to South Indian tunes. I have taken the liberty to insert some of his work in the example folder (".i.-gr.trial.mohanam;" and ".i.-gr.kss2;").

We recently came to know that a team located in Mumbai (= Bombay) had devised an excellent method for the .i.synthesis of tabla; sounds. We hope to make it accessible to BP2 users shortly.

**11. Programmed grammars**

When a generative grammar is used to derive a string, rule order is intrinsically predetermined by the availability of variables in the string under derivation. This process is generally non-deterministic because there may be several candidate rules. The idea of a .i.**programmed grammar**;, as suggested by Laske (1973a:365), is to impose an extrinsic ordering of rules reflecting a certain manner in which the generation process is envisaged by the composer. Alternate methods involve .i.**production procedures**; (see reference manual §8.1).

**11.1 Using flags**

Programmed grammars in BP2 use .i.**flags**; taking integer values. Suppose we need to generate random variations of length 6 containing 'a' and 'b', yet with exactly four occurrences of 'a' and two of 'b'. We may write (see ".i.-gr.tryflags;"):

RND

GRAM#1[1] S --> X X X X X X /flag1 = 4/ /flag2 = 2/

GRAM#1[2] /flag1 -1/ X --> a

GRAM#1[3] /flag2 -1/ X --> b

The first rule generates "X X X X X X" and six flags. More precisely, "flag1" is set to integer value 4 whereas "flag2" is set to 2.

These are used as conditions for the application of rules [2] and [3]: a rule may be applied only if the values of all its .i.condition flag;s are **strictly positive**. In addition, each time rule [2] (or [3]) is applied, "flag1" (or "flag2") is decremented by one unit.

Productions look like:

a a a a b b, b a a a b a, b a a a b a, a a b a b a, a b a b a a, b a b a a a, a b a a a b...

Another example: generate a string of length 10 on alphabet {a,b,c} in which 'b' and 'c' have equal numbers of occurrences (±1). The following grammar
(".i.-gr.tryflags2;") will do it:

[1] S --> X X X X X X X X X X /make\_b = 1/

[2] /make\_b -1/ X --> b /make\_c +1/

[3] /make\_c -1/ X --> c /make\_b +1/

[4] X --> a

This grammar produces:

c a b b b c c a a a, a b a a b a a c c b, b a b c a b b c c a, c a b a a a b a a c...

We recommend reading about .i.**production procedures**; (reference manual §8.1) that solve equivalent problems.

**11.2 New flag syntax**

From version 2.5 onward it is possible to write flag conditions as per the following examples:

/flag1 - 3/ X --> a [means that /flag1/ is to be decremented by three units]

/flag1 + 4/ X --> a [means that /flag1/ is to be incremented by four units]

The second rule is therefore increasing flags values. It could be equivalently written:

/flag1/ X --> a /flag1 + 4/

Many other new features on flags have been implemented in version 2.5. Here is a brief survey of .i.**flag conditions**; taken from the on-line help under "Flags":

/myflag = 3/ X --> a checks that 'myflag' is equal to 3, and if so, makes the rule candidate.

/myflag ≠ 3/ X --> a checks that 'myflag' is unequal to 3, and if so, makes the rule candidate.

/myflag < 3/ X --> a checks that 'myflag' is smaller than 3, and if so, makes the rule candidate.

/myflag > 3/ X --> a checks that 'myflag' is greater than 3, and if so, makes the rule candidate.

/myflag ≤ 3/ X --> a checks that 'myflag' is smaller or equal to 3, and if so, makes the rule candidate.

/myflag ≥ 3/ X --> a checks that 'myflag' is greater or equal to 3, and if so, makes the rule candidate.

Be careful that '=' is the **assignment** operator if found in the right argument of a rule, and a **comparison** operator otherwise. To get '≤' (inferior or equal) on a US keyboard, type option comma, and to get '≥' (superior or equal) type option period.

A rule may now contain several flag conditions, e.g.:

/myflag ≥ 3/ /myflag < 20/ /otherflag = 4/ X --> a

In assignments and comparisons, numbers may be replaced with other flags or control parameters K1, K2, etc., e.g.:

 X --> a /flag1 = K1/ /flag2 = flag1/

 /flag1 > K19/ X --> b

 /flag2 ≠ flag3/ X --> c

Be careful that all flags are set to zero when computation starts, unless the project is running in "Improvize" mode and "Reset rule flags" is not checked on dialog "Settings". In the latter case, flags keep the values they have acquired during the preceding computation.

**11.3 Classical problems solved with flags**

We now take an example showing that the new flag syntax provides elegant and comprehensive solutions to problems often encountered when dealing with formal grammars: (1) how do I control the global length of strings produced by a .i.self-imbedding grammar;? (2) how do I manage to place particular symbols at specific locations in all strings the grammar produces?

The example is project ".i.-gr.tryflags3;" and deserves comments.

// This grammar produces all strings of length 8 containing a's and b's, and c's in positions 4 and 6. Every string contains at least two a's.

RND

[1] S --> X /pos = 1/ /done = 0/

[2] X --> a X /pos +1/ /done +1/

[3] X --> b X /pos +1/

[4] <∞> /pos > 8/ X --> lambda [Infinite weight]

[5] <∞> /pos ≥ 7/ /done < 2/ X --> a X /pos +1/ /done +1/ [Infinite weight]

[6] <∞> /pos = 4/ X --> c X /pos +1/ [Infinite weight]

[7] <∞> /pos = 6/ X --> c X /pos +1/ [Infinite weight]

Rules [2],[3],[5],[6],[7] are **self-imbedding**, i.e. their left arguments are substrings of their right arguments. Consequently, should these rules remain candidate forever, the length of the work string would grow infinitely. So far, the only techniques to limitate the growth of the work string were dynamic weights (see §4.6 of reference manual) or limited buffer size (§4.7). The inconvenient of dynamic weights is that they make it difficult to control the probabilities of each candidate rule. Flags provide an independent (and numerical) control of derivations.

In the above example, flag 'pos' measures the length of the work string, which may not exceed 8 units. Rule [4] becomes compulsory (its weight being infinite) as soon as the flag condition pos > 8 is fulfilled.

Flag 'done' controls the number of occurrences of 'a', which must be minimum 2 in the final item. Rule [5] takes care of that condition: if less than 2 a's have been produced when the length is greater than 6, then this rule becomes compulsory and produces the missing terminals.

Rules [6] and [7] take care of the compulsion to place occurrences of 'c' in positions 4 and 6.

**12. Step computation**

So far we have seen how BP2 can be used to produce items randomly. BP2 makes decisions on candidate rules and positions of derivation following a (pre-set) pseudo-random sequence generated by the computer, as indicated in §5.1.

Another way of controlling computation is to make stepwise decisions on candidate rules. This becomes possible by selecting option ".i.Choose candidate rule;" (see Fig.31 infra). Now BP2 will display computation steps along with the rules used, prompting the user to click "Resume" at each step. When there are several candidate rules the user is prompted to mark the selected one.



**Fig.31 Prompting the user for a rule**

To select a rule just click on it, then click "Resume" or type cmd-r. If the "Resume-Undo-Stop" window disappears you may click the upper bar (the drag region) of the "Trace" window to bring it to front.

In the example below rule "X --> a" has been selected:



**Fig.32 Selecting a** .i.**candidate rule**;

If your decision leads to an unacceptable solution you may undo the computation typing cmd-z or selecting ".i.Undo;" in the "Action" menu. Undoing permits to backtrack as far as possible.

At any stage of the computation you may want to stop and save current choices. Click button ".i.Save decisions;". The current grammar is saved along with decisions. Decisions include the index of the subgrammar, the index of the selected rule and the position of the current derivation.

When loading a .i.decision file; (clicking ".i.Load decisions;") BP2 first checks whether or not the grammar currently in memory is the same one as the grammar to which decisions were refering. (BP2 remembers the date and time at which the grammar was compiled.) If the grammar has changed it suggests to delete the current grammar and replace it with the version saved along with the decision file.

Forcing a set of decisions on a grammar which is no longer compatible may produce unpredictable results *including crashing the system*.

Once a .i.decision file; is loaded, select ".i.Step-by-step compute;" and click ".i.Repeat computation;". When reaching the point at which the production had been interrupted, the user is prompted to continue making his/her own decisions.

**13. Manual time setting**

If ".i.Display time setting;" is selected (see Fig.24) BP2 offers you the option to display the timings of objects and participate in decisions regarding constraint satisfaction.

The time setting problem is introduced in §5.1 of the reference manual and discussed in Bel 1990b,1991,1992. The procedure described in §5.2 of the reference manual may be monitored with ".i.-da.checkTimeset;".

**14. Saving and loading MIDI code**

**14.1 MIDI files**

To allow the production of .i.MIDI file;s, type cmd-option space or select "Settings" in the "Windows" menu. Then check button "Write MIDI files". Whenever "Produce items" (cmd-r) or "Play selection" (cmd-p) are invoked, you are now offered to save the item to a MIDI file. The first time BP2 will prompt the creation of a MIDI file. Then it will propose to create a new file or go on saving to the current one. These are the default .i.**file saving options**; which can be modified in a dialog displayed with "Settings" (cmd-option space), see Fig.33.



**Fig.33 Default file saving options**

BP2 supports three .i.**MIDI file formats**;: types 0, 1 and 2. Here are the tips for choosing an output format:

• If the file will contain a single musical item, or a sequence of items produced by a grammar in the "Improvize" mode, any format is OK. Keep in mind, however, that format 2 may be rejected by single-track MIDI programs.

• In the current version, BP2 saves each item to a single track. Types 0 and 1 may therefore be considered equivalent.

• If the file will contain several independent musical items (each of which might have a different tempo or time signature), then type 2 must be used.

MIDI files may also be imported by sound-object prototypes (see §2.3.1).

**14.2 Storing and sending MIDI data**

".i.Receive MIDI data to file;", in the "File" menu, makes it possible to store any in-coming stream of MIDI messages to a text file. It may for instance be used to save the set-up of a MIDI device using its bulk transfer facility. Time information is not saved, but it is normally not required for system exclusive messages.

Conversely, ".i.Send MIDI data from file;" sends data from a text file back to a MIDI device. It may be used to initialise the device.

On old machines like my good old Mac II-ci, these procedures are too slow for loading and sending streams larger than the buffer of the in-built MIDI driver (30,000 bytes).

The same features are found in most commercial MIDI programs. With the HyperMIDI stack which was supplied with early version of BP2, the stack's buffer had to be set to a size larger than that of the MIDI stream, e.g. 120,000 bytes for a dump of Roland D-50.

**15. More about windows...**

**15.1 The "Scrap" window**

Typing cmd-o in the .i."Scrap" window; allows loading text files of any type. This window provides work space to edit grammars and data. For instance, another grammar may be loaded to the "Scrap" window and parts of it copied to the current grammar.

The .i."Trace" window; may equally be used for storing text. However, it is also used by BP2 for displaying error messages, and sometimes BP2 needs to clear it (with the user's permission).

Clicking the go-away box of the "Trace" window erases its content after prompting the user to save changes.

**15.2 Adjusting window sizes and locations**

.i.window size;Window sizes and locations are saved with the current settings. If the name of the settings file appears on top of the grammar or data, each time the grammar or data is loaded the same window environment is reset.

If the project was created on a larger monitor, BP2 checks the dimensions of the current screen and avoids dragging or resizing windows beyond screen boundaries.

Users of large monitors may want to change default positions and sizes taking full advantage of their large screen. Start BP2 (with no project), drag and resize windows, then save settings as a start-up file.

**15.3 BP2 editor;**

BP2 is basically a word processor (enhanced as *bol* processor, where 'bol' stands for a musical-speech symbol) using the WASTE text engine designed by Marco Piovanelli. Unlike BP2 versions before 7.5.2, window contents are not limited to 32,000 characters.

Basic operations: copy, cut, paste, find/replace are implemented in the usual manner. Undo (cmd-z) works more or less as expected in word processors.

Moving the text cursor is accomplished by clicking the mouse or hitting the four direction keys.

If the **option key** is down, left/right arrows move by one **word** while up/down arrows move by one **paragraph.** In word processing, a paragraph is a sequence of characters between two carriage returns; it is displayed on several lines if it is larger that the window width. These movements with the option key down are much more practical than the usual scrolling.

If both the **option** and the **shift key** are down, moving by one word or one paragraph extends the current selection accordingly.

The **command key** (cmd) may be used for jumping to the beginning or the end of a line (left and right arrows), and to the beginning or the end of a text window (up and down arrows).

These commands using arrows in combination with option, shift or command keys are identical to the ones in well-known editors like Alpha or Symantec® C++.

Any musical item, grammar rule or script instruction in BP2 must be written as one single paragraph. This is a problem when text warping provokes undesirable lay-outs (depending on the window width). Because of this, a .i.**special line break**; character is supplied: '¬'. To use the special line break, type 'return' with the 'option' key down. (The same symbol and keyboard sequence are used by HyperTalk and AppleScript.) When interpreting the input, BP2 pastes together lines separated with '¬'.

**Tabulations** in BP2 documents are displayed as ordinary spaces. Nevertheless, typing tabs instead of spaces may be useful for formatting the text in a word processor.

BP2 has another useful feature borrowed from Symantec: typing cmd-b allows you to check the "balance" of brackets () and {}. You may try it with .i.polymetric expression;s contained in the last subgrammar of ".i.-gr.Mozart;". Each stroke selects a higher bracket level. After several strokes you get a beep (if brackets are not balanced) or the entire window is selected, meaning that all expressions are balanced. Note however that a correct grammar (e.g. ".i.-gr.koto3;") may contain rules with unbalanced brackets.

In addition, BP2 allows users to 'type' notes from a MIDI keyboard (using cmd-j, see §1.4) or map the keyboard to predefined tokens (see §10.4).

**15.4 Remarks about remarks**

BP2 supports four styles of remarks:

• AppleScript® or HyperTalk® style (**in scripts only**): inserting "--" forces the compiler to ignore the next characters till the end of the current line. Example:

 Activate window Scrap -- this is an AppleScript style remark

• BP1 style: the remark is between square brackets. Example:

 S --> do4 re4 [This a BP1 style remark]

• old C style: the remark starts with "/\*" and ends with "\*/". It may contain several lines. Example:

 /\* This an old C style remark \*/

• new C style: any line starting with "//" is ignored. Example:

 // This is a new C style remark

**16. What's in a "-se." file?**

A "-se." .i.**settings file**; contains all information about the current environment of BP2, which does **not** include data found in the current "-gr." grammar, "-ho." alphabet, "-in." interactive codes, "-kb" keyboard encoding, "-gl" glossary and "-mi." sound-object files.

This includes the current start string, the current random seed, all button settings, the current time base, time resolution and quantization, graphic scale, MIDI filters, tuning, file save options, MIDI file type, window dimensions and locations, colour options and font sizes.

To save the current settings, select ".i.Save settings;" in the "File" menu. To create a start-up file, select ".i.Modify '-se.startup';". BP2 remembers where it found the startup settings file and it will store the new settings into it. (Perhaps it is wise to keep a copy of ".i.-se.startup;" before doing this.

A project-specific settings file must be saved in the same folder as the grammar, and a data-specific settings file must be saved in the same folder as the data it relates to. The name of the settings file is automatically inserted on top of the grammar and data windows, so that it may be later automatically reloaded.

Note that settings files produced by old versions of BP2 might not be fully compatible. Differences may appear in window positions and/or the settings of radio buttons on the "Settings" dialogs. After loading an old project it is recommendable to check its settings and save them again.

**17. Csound on BP2 -- a primer**

**17.1 What is Csound, how does it relate to BP2?**

Csound is a program for the design and transformation of digitised sounds. A public domain software currently developed at the MIT, it can be compiled to run on various platforms, notably Unix, NeXT and MacOS. The current PowerPC version may be retrieved from <ftp://notam.uio.no/pub/mac/audio/csound.ppc.sea.hqx>. The Csound front page is <http://www.leeds.ac.uk/music/Man/c\_front.html>.

Unlike hardware sound processors, Csound puts no limitation to the number of instruments and the complexity of their processes. Indeed, a sophisticated orchestra file would request a great deal of computation time, but it remains workable on small computers whenever speed is not critical.

With recent hardware enhancements (multiprocessing, RISC...) Csound has become fast enough for real-time synthesis, making it possible to adjust various sound parameters interactively. In other words, musicians can explore sound spaces rather than rely on speculations as to what a certain formula would yield. This shift from theory to practice is giving Csound a second life at a time computer users feel allergic to batch-processing techniques and dream of immediate feed-back and interactive systems.

Csound works with two input files, the "orchestra file" and the "score" file, both in comprehensive text format. This makes it easy to musicians to share a broad library of sound design "recipes", a sample of which is supplied with the software itself.

BP2 may create a piece of music in the real-time MIDI environment and then enhances it with Csound. Csound and BP2 are similar and complementary in their text-oriented, programming approach, with respective emphasis on sound and musical structures. None of these programs make assumptions on the musical system even though they optionally accept practical conventions such as "octave point pitch-class" representation in Csound, or "simple notes" in BP2.

A Csound instrument is the procedural representation of computational processes at the lowest level; conversely, a BP2 sound-object is a declarative representation of elementary events and properties at a more abstract level.

While Csound is quite user-friendly for the description of instruments (the "orchestra file"), it does not offer any in-built facility for the design of event lists (the "score" file). Score files are usually typed or produced with higher-level algorithms such as the ones programmed with .i.Cscore;, a utility supplied on the same site. But not every musician is a C programmer... A convenient way of generating Csound scores is to convert them from MIDI files, but this limitates Csound parameters to the very few ones implemented in MIDI. BP2 handles any number of parameters (with arbitrary names) that may be assigned any position among Csound instrument arguments.

(*Subject to confirmation by Csound folks*) BP2 is to-date the most advanced program for the creation of Csound score files. BP2's input may be comprehensive text scores (with an alphabet of simple notes and sound-objects) or music production processes such as grammars or scripts. BP2 attempts to take full advantage of the versatility of Csound, by-passing the limitations of MIDI.

In the following examples (except §17.5) we are using imaginary orchestra files and proposing a visual, rather than sonic, verification. The reason is that we intend to demonstrate a great variety of parameter formats.

**17.2 Producing Csound scores with BP2**

First select the ".i.Csound;" option in the "Output" menu. This option is saved with the settings of the current project or data. If the "MIDI" option is still active, musical items will be played on MIDI and then the user will be prompted to go for a Csound score. If "MIDI" is inactive Csound scores are immediately saved to a file. (See more options Fig.33.)

If Csound is running on a Unix or Windows machine it is a good idea to change the file format to "DOS" or "Unix". The dialog for MIDI and Csound file saving options is visible after typing cmd-option space ("Settings" in the "Windows" menu).

On the top "Settings" dialog, checking ".i.Trace Csound;" requests BP2 to display a copy of the score in the "Trace" window. A good idea for a first check-up.

BP2 creates a Csound score of musical items selected in a text window (a ".i.BP2 score;") or produced by a grammar. The components of these musical items are sound-objects and/or simple notes. A sound-object contains a sequence of elementary events that may be described as a MIDI stream, a Csound score, or both.

In the latter case, BP2 checks that the durations of the MIDI and Csound sequences are identical. It makes sense to have both descriptions generate similar sounds on MIDI and Csound, so that MIDI is used for an immediate check and Csound for the final material.

• When sound-objects contain Csound scores, Csound score in sound-object prototypetheir MIDI streams are ignored and the final score is a rearrangement of these score lines with appropriate changes in the timings of events and the values of their parameters. (See §17.8)

• When a MIDI stream is used as the input (which is always the case with simple notes), BP2 does its best to convert MIDI messages to Csound score lines. (See §17.3)

In both cases BP2 needs information about the parameter locations and ranges for each instrument of the Csound orchestra file. Unfortunately, reading the orchestra file would not yield the required information unless it is prepared in a comprehensive way. (Csound designers might think of new instructions in Csound orchestra files that would make this information directly accessible :-) This partial description of Csound instruments is entered in the "Csound instruments" dialog and saved to a ".i.-cs.<name>;" file.

Version 2.9.3 of BP2 inserts function tables into the score when necessary, thus enabling arbitrary variations of continuous parameters within a single Csound event (a score line).

**17.3 Try Csound with simple notes**

Select the "Data" window and type cmd-o to load "-da.tryCsound" in the examples folder. On top of the "Data window" the instruction

-cs.tryCsound

tells BP2 to immediately load the ".i.-cs.tryCsound;" file describing the orchestra. Since no ".i.-mi.<name>;" is attached, musical items will contain only simple notes, and Csound scores will be based on the conversion of MIDI messages.

Select for instance the sequence of simple notes

 A4 G4 C5 A4

and type cmd-p (".i.Play selection;"). This item is played on MIDI and immediately saved to a Csound score file containing:

t 0.000 60.000

i2 0.000 1.000 440.00 90.000 0.000 0.000 0.000 ; A4

i2 1.000 1.000 392.00 90.000 0.000 0.000 0.000 ; G4

i2 2.000 1.000 523.25 90.000 0.000 0.000 0.000 ; C5

i2 3.000 1.000 440.00 90.000 0.000 0.000 0.000 ; A4

s

Each line starts with "i2" meaning that instrument 2 is used. BP2 found this value in the settings of instrument 2:

Assign as default to channel 1

and MIDI channel 1 is the one used by default.

On each score line, the "2" of "i2" is the value of argument 1, the leftmost one. The next two arguments are always assigned to the on-setting date and duration of the event, measured in beats. Then comes argument 4, which in this example is assigned to pitch, here in cycles per second. Argument 5 is the volume (range 0..127) set to 90 by default. Arguments 6, 7 and 8 are used for another parameter not demonstrated here.

Instrument 2 does not accept any other parameter. Therefore, velocity, pressure, modulation, etc., assignments are ignored.

**17.4 Describing a Csound instrument**

Look at the .i."Csound instruments" dialog; accessible from the .i."Output" menu;. Instrument 3, which (for fun) we named a "Harpsichord", is shown Fig.34.



**Fig.34 Describing a Csound instrument in the BP2 environment**

This instrument assigns pitch to argument 4 but it uses the "cps (Hz)" frequency representation. Further, it assigns pitchbend to arguments 5 and 6. Why two arguments? Since pitchbend may vary continuously, it is smart to supply the values at the on-setting and off-setting dates so that Csound may do something with it. There is an empty square to the right of these arguments, just after the question mark. It may contain another argument index for passing on tabulated functions to Csound when movements become more complex. (Since BP2 does it automatically, you don't have to bother about understanding Csound function tables.)

In MIDI a NoteOn starts a process, a NoteOff terminates it, and between the two the sound generation may be modified in real time by control messages. But in a Csound event all arguments are supplied in advance. Therefore, it is the task of the instrument procedure to modify its own parameters. If only start and end values of the control parameter are supplied, Csound has no other option than perform a simple interpolation. However, with sound-objects additional information may be passed on to Csound (see §17.11).

Arguments 7 and 8 of this bizarre harpsichord are used similarly for continuous volume control. In addition, MIDI values in range 0..127 are mapped to the range -24..24, which is declared a "logarithmic" scale. Here we assumed that "Harpsichord" would takes decibels as a volume input value. Since performance control \_volume(x) uses the MIDI range 0..127, a remapping is automatically done by BP2 on the basis of three supplied pairs of values.

A similar mapping is applied to modulation (arguments 11 and 12) and panoramic (argument 13) parameters. Since the latter has only one argument it cannot be used in continuous variation with this instrument.

Csound argument mapping is explained in BP2 reference manual (§9.1). All you need to know is that the first line contains three input values and the second one the three corresponding output values. BP2 manages to find a proper quadratic and/or linear interpolation matching these points. Error messages are displayed in rare cases when the mapping is impossible.

The input, output, or both input and output scales of the mapping may be declared as "logarithmic" for a more meaningful conversion.

Let us for instance produce a Csound score for the musical item:

\_ins(Harpsichord) \_volumecont \_volume(127) A4 G4 C5 A5 A4 G4 C5 A5 \_volume(0)

which specifically invokes instrument 3, using its name, and allows the volume to vary continuously from 127 to 0. The Csound score is:

t 0.000 60.000

i3 0.000 1.000 440.00 8191.500 8191.500 24.000 21.688 0.000 0.000 0.000 0.000 0.000 ; A4

i3 1.000 1.000 392.00 8191.500 8191.500 21.688 18.431 0.000 0.000 0.000 0.000 0.000 ; G4

i3 2.000 1.000 523.25 8191.500 8191.500 18.431 12.899 0.000 0.000 0.000 0.000 0.000 ; C5

i3 3.000 1.000 880.00 8191.500 8191.500 12.899 -0.031 0.000 0.000 0.000 0.000 0.000 ; A5

i3 4.000 1.000 440.00 8191.500 8191.500 -0.031 -1.179 0.000 0.000 0.000 0.000 0.000 ; A4

i3 5.000 1.000 392.00 8191.500 8191.500 -1.179 -2.794 0.000 0.000 0.000 0.000 0.000 ; G4

i3 6.000 1.000 523.25 8191.500 8191.500 -2.794 -5.547 0.000 0.000 0.000 0.000 0.000 ; C5

i3 7.000 1.000 880.00 8191.500 8191.500 -5.547 -24.000 0.000 0.000 0.000 0.000 0.000 ; A5

s

The column of argument 4 (pitch) is self-explanatory. Arguments 5 and 6 contain the constant default value of pitchbend (8191.5). Arguments 7 and 8 are the ones showing the volume decreasing from +24 to -24. Arguments 11, 12 and 13 keep their default values. Arguments 9 and 10 are used by another parameter (see 'blurb' infra) here set to 0.

**Important**

If you use the "cps (Hz)" frequency representation, do not forget to "tune" BP2 in case the .i.**diapason frequency**; is not 440Hz. (See §4.8.)

So far we demonstrated that parameters mapped to MIDI controls, i.e. pitchbend, modulation, volume, panoramic and pressure, can be controlled continuously and mapped to Csound parameters. More parameters, specific to Csound, may be defined in the dialog accessed clicking the "MORE..." button (see Fig.34). There is room for six additional parameters in each Csound instrument, a total 250 in the entire orchestra.



**Fig.35 Description of** .i.**additional Csound instrument parameters**

Each additional parameter has a default value which is not necessarily zero. Values found on Csound scores may be combined additively (.i.ADDval;) or multiplicatively (.i.MULTval;). Tables describing fine variations of the parameter may use arbitrary Csound generators. .i.GEN07; (linear interpolation) is assigned by default, but .i.GEN08; (cubic spline) may be prefered in some cases.

Additional parameters are given arbitrary names. They are controlled by instructions ".i.\_value;(param,x)", ".i.\_step;(param)", ".i.\_cont;(param)" and ".i.\_fixed;(param)". An example in ".i.-da.tryCsound;" demonstrates the control of an arbitrary parameter labelled 'blurb' which is recognized by both instruments "Harpsichord" (3) and "Flute" (2).

\_ins(Harpsichord) \_cont(blurb) \_value(blurb,123.42) C4 D4 \_ins(Flute) E4 F4 \_value(blurb,-211) \_step(blurb) G4 A4 \_value(blurb,-34) \_fixed(blurb) B4 C5

i3 0.000 1.000 261.63 8191.500 8191.500 16.981 16.981 123.420 39.815 0.000 0.000 0.000 ; C4

i3 1.000 1.000 293.66 8191.500 8191.500 16.981 16.981 39.815 -43.790 0.000 0.000 0.000 ; D4

i2 2.000 1.000 329.63 90.000 -43.790 -127.395 0.000 ; E4

i2 3.000 1.000 349.23 90.000 -127.395 -211.000 0.000 ; F4

i2 4.000 1.000 392.00 90.000 -211.000 -211.000 0.000 ; G4

i2 5.000 1.000 440.00 90.000 -122.500 -122.500 0.000 ; A4

i2 6.000 1.000 493.88 90.000 -34.000 -34.000 0.000 ; B4

i2 7.000 1.000 523.25 90.000 -34.000 -34.000 0.000 ; C5

Additional parameters take any values in floating-point format, but they cannot be remapped before being sent to Csound instruments (see reference manual §9.1 regarding remapping). The use of .i.**control parameter**;s K1, K2... is not licit in the current version of BP2.

**17.5 Checking the .i.Csound output**

A Csound orchestra file named ".i.BP2test.orc;" is supplied. It works with any Csound score produced by BP2 in the absence of a Csound instrument description file (.i.-cs.<name>; file).

Below is the listing of the .i.**default orchestra file**;.

; This is a simple orchestra file for BP2's default Csound score output

; It follows the new specfications handled by BP2 version 2.7.2 and above.

; (See "Csound tables")

; It contains one single instrument: an oscillator using wave-table 1

; Argument p2 is the start date, p3 the duration (as per the standard)

; Argument p4 is the pitch in octave point pitch-class format

; Volume (range 0..127) is supplied by performance control \_volume()

; to arguments p5 (beginning value) and p6 (end value), or via the table

; whose index is supplied by p7.

; Pitchbend is supplied in cents to arguments p8 (beginning value)

; and p9 (end value), or via the table whose index is supplied by p10.

 sr = 22050

 kr = 2205

 ksmps = 10

 nchnls = 1

 instr 1

ik1 = 32767. / 127.

ik2 = log(2.) / 1200.

ifvol = p7

ifcents = p10

kvol line p5, p3, p6

if (ifvol <= 0) goto volumelin

ilenvol = ftlen(ifvol)

kndxvol line 0, p3, ilenvol

kvol tablei kndxvol, ifvol

volumelin: kcents line p8, p3, p9

if (ifcents <= 0) goto pitchbendlin

ilencents = ftlen(ifcents)

kndxcents line 0, p3, ilencents

kcents tablei kndxcents, ifcents

pitchbendlin: kpitch = cpspch(p4) \* exp(kcents \* ik2)

kamp = kvol \* ik1

a1 oscil kamp, kpitch, 1

 out a1

 endin

This code should not look esoteric to Csound users. The unique instrument in this orchestra is an oscillator controlled in volume by arguments 5 and 6 (start and end respectively) and in pitch by argument p4 (octave point pitch-class format). Pitchbend corrections are given in cents by arguments 8 and 9 (start and end respectively). In addition, arguments 7 and 10 may contain the references of function tables when complicated variations are described.

We are confident that Csound users will pick up the idea and develop their own instruments taking advantage of BP2 features such as panoramic control, and any additional control they think useful. We hope that convincing examples will be shared between users.

Type cmd-n ("New Project") to flush the current Csound orchestra description and use the default instrument exclusively. Then load ".i.-da.Csoundtest;" in the "Data" window.

\_volumecont \_pitchcont \_pitchrange(200) \_volume(30) C5 \_volume(127) D5 \_pitchbend(100) E5 F5 \_pitchbend(0)

Select this item and type cmd-p ("Play selection"). Save the resulting score as "BP2test.sco":

; Csound score

f1 0 256 10 1 ; This table may be changed

t 0.000 60.000

i1 0.000 1.000 9.00 30.000 127.000 0.000 0.000 0.000 0.000 ; C5

i1 1.000 1.000 9.02 127.000 127.000 0.000 0.000 0.000 0.000 ; D5

i1 2.000 1.000 9.04 127.000 127.000 0.000 100.000 50.000 0.000 ; E5

i1 3.000 1.000 9.05 127.000 127.000 0.000 50.000 0.000 0.000 ; F5

s

e

; this score was created by Bol Processor BP2 (version BP2.7.2)

To close the score file, type cmd-option space (window "Settings") and uncheck ".i.Write csound scores;". Now you may open the score with Csound. (If the computer does not have enough memory, first quit BP2.)

Tell Csound to create a sound file using ".i.BP2test.orc;" and ".i.BP2test.sco;". Listen to the result. (Nothing great. It just works.)

If you are not happy with the buzz you may try to change the waveform in table 1, as explained in §17.6. Apart from being extremely dull, a sine-wave oscillator may yield weird results when several notes are superimposed. Since BP2 provides Csound with events timed within 1ms accuracy, phasing effects modify considerably the perceived balance of mixed sounds.

Beware of computation overflows occuring at the time Csound processes the score file. These are displayed at computation time. It is wise to use \_volume(x) with a proper value of 'x' in range 0..127 so that saturation is avoided. A simple method consists in dividing the maximum volume (127) by the estimated number of simultaneous voices, and use the result as a maximum volume for each voice. Use .i.**control parameter**;s \_volume(K1), \_volume(K2)... to avoid repeating numeric data (see §1.9).

**.i.17.6 Csound function tables describing waveforms**

Most Csound instruments need tables to produce wave-forms, ramps, etc. These tables are stored by BP2 along with the -cs.<name> file and may be accessed clicking the top left button "Tables..." on the Csound instrument dialog (see Fig.34). Default table "f1" is written following the syntax of Csound:

f1 0 256 10 1

It contains 256 points and uses GEN10 (the harmonic oscillator) with only the first partial, yielding a sine-wave. This is the default table used by BP2.

.i.Csound waveform description;s are automatically inserted in the beginning of Csound score files. They are not displayed in the "Trace" window even

when ".i.Trace Csound;" is active. Do not use indexes beyond 100 as these are reserved to function tables automatically produced by BP2 (see §17.12 infra).

**17.7 Sound-objects containing Csound scores**

Sound-objects contain sequences of elementary events which may be described as a MIDI stream, a Csound score or both.

When BP2 is instructed to produce MIDI real-time or a MIDI file, it uses the MIDI stream of sound-objects and ignores their Csound definitions. Conversely, when asked to write the performance as a Csound score, it uses the Csound score of each sound-object. If the sound-object has no score, BP2 converts its MIDI stream to a Csound score, using the current or default Csound instrument description, as shown in the preceding section.

To produce Csound scores it is wiser to design .i.Csound scores in sound-object prototypes; rather than rely on a conversion from MIDI which may leave out relevant parameters.

A demo of the use of sound-objects containing Csound scores is given in ".i.-da.tryCsoundObjects;". Type cmd-e ("Edit prototypes") to load the attached ".i.-mi.tryCsoundObjects;" sound-object prototype file. Then click button "Csound" to display the Csound score of prototype 'a' (see Fig.36).


**Fig.36 The Csound score of a sound-object prototype**

The score contains four events invoking instruments 1, 2 and 3. Note that this score is incomplete because it contains only the compulsory arguments for setting time, duration and pitch. BP2 will complete missing arguments with their default values.

To compile this score, BP2 requires the description of instruments supplied by the ".i.-cs.tryCsoundObjects;" Csound instrument file. This file has been loaded automatically for two reasons:

• Its name appears on top of the ".i.-da.tryCsoundObjects;" file

• Its name has been found in the ".i.-mi.tryCsoundObjects;" file (on the left top of the window, see Fig.36). It can be changed by clicking the ".i.Change instrument file;" button.

To avoid inconsistencies it is sufficient to state the name only once, preferably in the sound-object prototype file.

On the graphics Csound events are marked with red lines whereas MIDI events show as black lines. Objects 'a', 'b' and 'c' in this file contain only Csound scores (as told by radio buttons on Fig.36), whereas the sound-object prototype 'midiobject' also contains a MIDI stream. The MIDI stream and the Csound score of an object may contain similar events as to pitch and timing, but each of them may also contain additional information that is specific to the representation: Csound score lines contain parameters that have no equivalent in the MIDI stream, and conversely a few MIDI messages may not be reflected in a Csound score.

BP2 checks that the Csound score and the MIDI stream, if not empty, represent sequences of events with identical durations. If this is not the case a message is displayed although the user is not forced to correct the mismatch. In this case, the longest duration is taken as the nominal duration of the sound-object prototype.

**17.8 Creating a Csound score in a sound-object prototype**

Csound score in sound-object prototypeThere are three ways of creating a Csound score describing the sequence of events in a sound-object prototype.

• Type the score in the window displayed Fig.36;

• Import the score from a text file (which could as well be a score produced by BP2);

• Create the score by converting the MIDI stream to Csound.

The third method is a practical way of converting BP2 projects from MIDI to Csound. First of all, a Csound instrument file such as ".i.-cs.tryCsoundObjects;" must be loaded so that the conversion makes use of the proper instruments. Then, for each sound-object prototype, click the button ".i.Convert to Csound;". If necessary, the MIDI stream may be deleted afterwards: click ".i.Edit/record MIDI stream;" and "CLEAR".

Needless to say, sound-objects containing a Csound score may be granted the same properties: pivot, continuity, periodicity (§2.5) etc., as the ones containing a MIDI stream. Thus, Csound users can benefit of advanced features of BP2 as to timing, constraint satisfaction, etc.

**17.9 Performing musical items in the Csound environment**

For instance we request BP2 to perform

{a,b} c

with sound-object prototypes a, b, c containing the following Csound scores:

Score for a

t 0 120

i1 0 0.5 4.05 ; F0

i1 0 0.5 4.05 ; F0

i2 1.5 0.5 5.05 ; G5

i3 1.5 0.2 643.5 1 ; D#5

e

Score for b

t 0 60

i3 0 0.8 461.34 1; A4

i4 0.1 0.2 6.03 1.05 ; C2

e

Score for c

t 0 120

i2 0 0.2500 9.00 0 0 90 90 ; C5

i2 0.2500 0.2500 9.02 0 0 62 62 ; D5

i2 0.500 0.2500 9.04 0 0 90 90 ; E5

i2 0.7500 0.2500 9.05 0 0 90 90; F5

e

Properties of these sound-objects are the default ones except for 'b' which has its pivot delayed 20% of its duration

The final score (see below) is a combination of the prototype scores shown above. It reads:

t 0.000 60.000

i3 0.000 0.800 461.34 1.000 ; A5

i4 0.100 0.200 6.03 1.050 ; C2

i1 0.160 0.250 4.05 90.000 90.000 0.000 0.000 0.000 0.000 ; F0

i1 0.160 0.250 4.05 90.000 90.000 0.000 0.000 0.000 0.000 ; F0

i2 0.910 0.250 5.05 0.000 0.000 90.000 90.000 0.000 0.000 0.000 ; F1

i3 0.910 0.100 643.50 1.000 ; D#6

i2 1.160 0.125 9.00 0.000 0.000 90.000 90.000 0.000 0.000 0.000 ; C5

i2 1.285 0.125 9.02 0.000 0.000 62.000 62.000 0.000 0.000 0.000 ; D5

i2 1.410 0.125 9.04 0.000 0.000 90.000 90.000 0.000 0.000 0.000 ; E5

i2 1.535 0.125 9.05 0.000 0.000 90.000 90.000 0.000 0.000 0.000 ; F5

s

BP2 appends note names as remarks to events that invoke a pitch-sensitive instrument: 'A5', 'C2'...

Note the changes on timings in the score lines borrowed from sound-object prototypes 'a' and 'c'. These objects have been performed at tempo mm = 60 while their own reference tempo was mm = 120. Since Csound represents time in numbers of beats, their relative durations have been divided by 2.

It may look abnormal that all on-setting dates (except for the first two events) have been delayed by 0.160 beat. The explanation lies in the time setting of the musical item recalling the particular pivot location of sound-object 'b' (see Fig.37).



**Fig.37 The time setting of "{a,b} c"**

**17.10 Performance controls and Csound scores**

Other examples in ".i.-da.tryCsoundObjects;" demonstrate modifications of Csound argument values caused by performance controls. Try for instance:

\_pitchrange(200) \_pitchcont \_volumecont \_volume(30) \_pitchbend(0) a C4 D4 b c \_pitchbend(100) \_volume(127)

containing sound-objects 'a', 'b', 'c' and simple notes 'C4' and 'D4'. Simple notes are set on the default MIDI channel 1 which is also assigned instrument 1 as shown in ".i.-cs.tryCsoundObjects;".

The graphic score is shown Fig.38.



**Fig.38 The time setting of "a C4 D4 b c"**

The different instruments called to perform the resulting sequence of events have various ways of dealing with pitchbend and volume controls. For instance, instruments 3 and 4 do not control pitchbend continuously but they take the current value as argument 5 converted to frequency ratios in particular ranges. Instruments 1 and 2 accept continuous pitchbend controls expressed in cents. Volume is only taken in consideration by instruments 1 and 2.

The output score looks clumsy because of the diverse argument formats used by instruments for this demo.

t 0.000 60.000

i1 0.000 0.250 4.05 30.000 34.850 0.000 -10.000 -4.500 0.000 ;F0

i1 0.000 0.250 4.05 30.000 34.850 0.000 -10.000 -4.500 0.000 ;F0

i2 0.750 0.250 5.05 6.499 11.999 44.550 49.400 0.000 0.000 0.000 ;F1

i3 0.750 0.100 643.50 1.005 ;D#6

i1 1.000 1.000 8.00 49.400 68.800 0.000 12.000 34.000 0.000 ;C4

i3 2.840 0.800 461.34 1.069 ;A5

i4 2.940 0.200 6.03 1.151 ;C2

i1 2.000 1.000 8.02 68.800 88.200 0.000 34.000 56.000 0.000 ;D4

i2 4.000 0.125 9.00 77.990 83.490 107.600 112.450 0.000 0.000 0.000 ;C5

i2 4.125 0.125 9.02 83.490 88.989 77.466 80.807 0.000 0.000 0.000 ;D5

i2 4.250 0.125 9.04 88.989 94.488 117.300 122.150 0.000 0.000 0.000 ;E5

s

Incidentally, this example shows that BP2 does not necessarily produce a Csound score sorted on the on-setting dates of events. This is not a limitation since Csound sorts out dates after reading the score. BP2 does the same before compiling the Csound scores of its own sound-object prototypes.

**17.11 Combining argument values**

Let us have a closer look at the Csound score of sound-object prototype 'c'.

t 0 120

i2 0 0.2500 9.00 0 0 90 90 ; C5

i2 0.2500 0.2500 9.02 0 0 62 62 ; D5

i2 0.500 0.2500 9.04 0 0 90 90 ; E5

i2 0.7500 0.2500 9.05 0 0 90 90; F5

e

The last two arguments represent volume. Let p0 be the value in a particular place of a score line. (For instance, p0 = 62 on the second event.) If the volume is not explicitly controlled, then p0 is copied to the output score. However, if a "\_volume(x)" instruction precedes the sound-object, the output value p1 is calculated as follows:

p1 = p0 \* (x / default)

where 'default' is the default volume, i.e. 90 in BP2.

For other controls such as pitchbend, modulation, channel pressure and panoramic, a different formula is used:

p1 = p0 + x - default

where 'default' takes the following values:

• 8191.5 for pitchbend

• 0 for channel pressure and modulation

• 64 for panoramic

These calculations are performed in the MIDI parameter range. If the parameter is mapped to a different range in the Csound domain, p0 is first converted to the MIDI range (using the reverse mapping), the formula is applied, and then the result is mapped back to the Csound range.

**17.12 Tabulated functions**

In § 17.10 we demonstrated continuous changes of pitchbend and volume ranging over several sound-objects and simple notes. Changes were simple in that their variations could be described as two straight lines. Therefore the only useful information required by the Csound instruments were the start and end values of each parameter.

The situation is often more complex. Parameters may change following arbitrary definitions between the on-setting and the off-setting of a Csound event (a line of the score). BP2 handles this case automatically and builds a function table tied to the event which it inserts before the score line.

In ".i.-da.tryCsound;", try for instance:

\_ins(1) \_pitchrange(200) \_pitchbend(0) \_pitchcont C4 \_ \_pitchbend(20)\_ \_ \_pitchbend(165) \_ \_ D4 \_ \_ \_ \_pitchbend(150)

which yields the Csound score:

t 0.000 60.000

**f101** 0.000 256 -7 **0.000** 85 **20.000** 85 **165.000** 171 **160.000**

i1 0.000 6.000 8.00 90.000 90.000 0.000 **0.000 160.000 101.000** ; C4

i1 6.000 4.000 8.02 90.000 90.000 0.000 **160.000 150.000** 0.000 ; D4

s

Here the variation of pitchbend on 'D4' is simple and requires only the start/end values (bold on the score). However, the pitchbend variation on 'C4' requires four values: 0, 20, 165 and 160, which BP2 stores in the 'f101' function table linked to 'C4'. The table index '101' appears as the last argument in event 'C4'. When it is 0 (the case with 'D4') no function table is attached.

In ".i.-da.tryCsoundInstruments;" a few examples demonstrate BP2's ability to combine calculated parameter values with predefined ones contained in the Csound score of a sound-object. See for instance a complex variation of 'blurb' on sound-object 'e' which contains 4 lines of Csound score.

\_cont(blurb) \_value(blurb,12) e \_value(blurb,110) \_ \_value(blurb,130) \_ \_ \_value(blurb,80) \_ \_ \_value(blurb,-20) \_ \_value(blurb,-10) \_ \_ \_ \_value(blurb,40)

t 0.000 60.000

f101 0.000 256 -7 12.000 76 110.000 76 130.000 180 96.750

i2 0.000 3.330 7.04 8191.500 8191.500 64.000 64.000 12.000 96.750 101.000 ; E3

f102 0.000 256 -7 12.000 51 110.000 51 130.000 102 80.000 154 30.000

i2 0.000 5.000 7.00 8191.500 8191.500 64.000 64.000 12.000 30.000 102.000 ; C3

f103 5.000 256 -7 **40.000** 51 -6.000 51 8.000 205 **70.000** ; table shifted

i2 5.000 5.000 7.02 8191.500 8191.500 64.000 64.000 **40.000 70.000** 103.000 ; D3

f104 6.660 256 -7 -13.400 26 -10.000 230 39.917

i2 6.660 3.335 7.07 8191.500 8191.500 64.000 64.000 -13.400 39.917 104.000 ; G3

s

BP2 created function tables attached to each event. But there is more than that. In the Csound score of 'e', event 'D3' had non-zero initial values on arguments 9 and 10. These values have been added to the ones determined by variations of the 'blurb' parameter, thus yielding 40 for argument 9 (the start value of 'blurb') and 70 for argument 10 (the end value). The same modification is reported in table 'f103', and all values in the table changed accordingly.

The first and last values of a function table always reflect the start and end values of the parameter it describes variations of. This allows Csound instruments to operate consistently, either interpolating start and end values, or looking at the attached function table.

**17.13 Function tables in sound-object scores**

The current version of BP2 does not accept function tables in sound-object scores. This feature will be developed shortly. Function tables in sound-objects should be combined with the ones created by parameter variations in the performance.

**17.14 A diagram of the cooperation between BP2 and Csound**



**Fig.38 BP2 and Csound: two complimentary environments...**

BP2 Reference Manual

**Version 2.9.3 (Revised: February 1998)**

*The development of more and more visual stuff curtails the possibility of "thinking in your chair." Sometimes I develop grammars, not at the computer, but sitting with a pencil and paper. With programs [other than BP2] this is not possible: you must sit in front of the computer. The difference lies in the type of attention that each software environment demands on the part of the composer, and indeed reflects on the way s/he thinks about music.*

Harm Visser (in Bel 1998)

To understand the philosophy of Bol Processor BP2 in great depth it is recommended to read our publications: Kippen & Bel 1992, Bel & Kippen 1992, Bel 1992a-b. (Bel 1991) is a regularly updated document containing detailed descriptions of algorithms and data structures used in BP2.

Most examples described in these publications are enclosed in the "BP2 examples" folder.

**1. Task environment**

The earlier version of Bol Processor, namely BP1, was built on a model of "pattern grammars" enabling both the production and fast parsing of sentences (Bel 1987a-b). The grammar format in BP2 is much less restrictive as far as production is concerned. A sound interface has been implemented (using the MIDI standard) which is not geared towards any particular synthesiser; it accepts musical input from any MIDI device.

Below is a block diagram showing the interaction of modules in BP2:


**Fig.1 A block diagram of Bol Processor BP2**

Three fields are used for storing **grammars**, **items** generated by the .i.**inference engine**; and .i.**sound-object prototype**;**s**. Items are represented as structures (strings and sets) of symbols. Each symbol is mapped to a single sound-object prototype or a note. The interpreter generates MIDI codes or Csound score lines given the symbolic structure of an item and properties of sound-object prototypes.

Interpretation is in two stages. Musical items (which may be polyphonic) are represented as strings of symbols in a syntactic form called a .i.**polymetric expression**;. First, a mapping is calculated between the sound-objects contained in a polymetric structure and a set of **symbolic dates**. .i.Symbolic time;, here, is an arbitrary ordered set that permits the ordering of sound-objects. The mapping of symbolic to physical time is called the .i.**interpretation of** .i.**polymetric expression**;**s**. In this process, missing information regarding the ordering of sound-objects (along symbolic time) is inferred (Bel 1991,1992a). Then BP2 proceeds to the .i.**time setting**; of the sound-object structure represented as a .i.complete polymetric expression;. Start/clip dates of all sound-objects are calculated, yielding the dates of MIDI messages or Csound events. Sound-object properties are taken into account during time setting only.

The block diagram indicates that an external control can be exerted on the inference engine, grammars and the interpretation module. Specific MIDI messages may be used to change .i.weights in grammar;s, the time base and

nature of time (striated/smooth). These messages may also be used for synchronising events during their performance and even assigning computation time limits. Such features are currently used in improvisational rule-based composition.

Several BP2's may be linked together and to other software devices such as MIDI sequencers. Messages on the different MIDI channels and Apple Events may be used for communicating between machines or for controlling several sound processors. It must be kept in mind that a "sound-object" is not necessarily a sound-generating process. Depending on the implementation it may contain any kind of control/synchronisation messages as well.

**2. Technical features**

**2.1 File description**

(See QuickStart §3.1-ff)

We recommend using a presentation by names rather than large icons, so that files appear in alphabetical order and are sorted out on their prefixes.

The complete list of file types and creators used by BP2 is found in the on-line documentation under "Types-creators".

**2.2 File folders**

All files used by the same project, or their aliases, should be placed in the same folder.

"BP2 help" and "-se.startup" must be placed in the same folder as BP2.9.3.

**3. Sound-object prototypes**

(A formal introduction to sound-objects may be found in Bel 1991,1992a)

A string of terminals and structural symbols generated by the Bol Processor is converted to MIDI messages in the following way: each terminal represents a sequence of MIDI codes or a ".i.**musical gesture**;" recorded from a MIDI instrument. We call these sequences .i.**sound-object prototype**;**s**.

Sound-objects are generally not performed at the same speed they have been defined. Their time-scale depends both on the tempo of the performance and on certain properties assigned to their prototypes.

**3.1 Metrical properties of sound-objects**

The position of an object in relation to .i.time streak;s is calculated on the basis of some properties introduced now. We use the names of properties as per previous publications. These appear more explicitly in the new BP2 interface.

Property .i.**PivBeg**; means that the streak position is the first MIDI message of the object prototype. .i.**PivEnd**; means the last message. .i.**PivCent**; means the centre. .i.**PivBegOn**; means the first NoteOn. .i.**PivEndOff**; the last NoteOff. .i.**PivCentOnOff**; means the centre of the time-span interval starting on the first NoteOn and ending on the last NoteOff. .i.**PivSpec**; is a general case in which

the time offset between the first MIDI message and the streak is specified. These properties are explicitly shown in the dialog Fig.2.



**Fig.2 Editing metrical sound-object properties**

Fig.2 shows that the pivot location may either be defined absolutely (in milliseconds) or as a percentage of the object's duration. In the latter case, it will be modified according to the current time base of the performance if the object has properties OkRescale, OkCompress or OkExpand.

.i.**OkRescale**; means that this object may be expanded or compressed according to the time base. .i.**FixScale**; means it can't. .i.**OkCompress**; means it can be compressed, .i.**OkExpand**; that it can be expanded. (See explicit representation on Fig.2) Indeed, if OkRescale is true, then all other properties except FixScale are made true. The scaling of an object is expressed by the .i.**dilation ratio**; α. If α > 1 then the sound-object is performed slower than its prototype. The dilation ratio may be confined to a range deemed acceptable for the sound-object (see Fig.2). .i.Periodical sound-object;s are handled in a different way, see §2.5.

The value of α may be sent to the MIDI device (before the sound-object is performed) in case the sound processor is able to make sense of it for modifying parameters in synthesis algorithms. (Most commercial synths don't.) The current value of α is encoded logarithmically in range 0.01 to 100, and the MIDI controller value is



so that α = 0.01 will yield 0 and α = 100 yields 128. If you want this value to be passed to the sound processor each time this particular sound-object is performed, then check ".i.Send dilation ratio to controller;".

"Relocate at will" (.i.**Reloc**;) means that the pivot of the object may be located far from the time streak if necessary. "Never relocate" forces location on the time streak. Relocation may also be allowed within specific limits (see Fig.2).

**3.2 Topological properties of sound-objects**



**Fig.3 Editing topological sound-object properties**

.i.**OverBeg**; means that the beginning of the sound-object may overlap the end of an object preceding it in the same sequence. .i.**OverEnd**; means that its end may overlap another object following it in the same sequence. .i.**TruncBeg**; means that the beginning may be truncated if it overlaps an object in the same sequence. .i.**TruncEnd**; means that the end may be truncated if it overlaps an object in the same sequence. These properties (see Fig.3) may also be assigned within absolute or relative limits.

.i.**BrkTempo**; means that the object may "break" the striated structure (*organum*).

Continuity properties are shown Fig.4. .i.**ContBeg**; means that the beginning of this object should join (or overlap) the end of a preceding one in the same sequence. .i.**ContEnd**; means the end of this object should join (or overlap) the beginning of a following one. These properties may be compensated with a tolerance for a gap preceding or following the object.



**Fig.4 Editing continuity and MIDI properties**

.i.**MIDI properties**; (see Fig.4) instruct BP2 how to deal with channel, transposition, articulation, volume and panoramic changes as far as this object is concerned.

Given these sets of properties (stored in the "-mi." file) and a setting of the time base, BP2 will try to set all objects in time. For this it may need to relocate some of the objects that have Reloc or BrkTempo properties, truncate other objects, etc. If there is no solution, BP2 may release some of the constraints. (See operational mode §5.2)

**4. A short glance at the theoretical model**

**4.1 Variables and terminal symbols**

A .i.**terminal symbol**; is a symbol to which a single .i.**sound-object**;may be assigned. Terminal symbols are either predefined (see QuickStart §1.4: .i.**simple note**;**s**) or enlisted in an .i.**alphabet file**; (with prefix "-ho.").

Terminal symbols always start with a lower-case character a..z, or a character in the set

ÄÅÇÉÑÖÜáàâäãåçéèêëíìîïñóòôöõúùûüAEaeøÀÃÕOEoeÿßØµ∂∑∏πΩƒ∆¥#•

and may contain any character in the above set or in the additional set:

0123456789#%\*@¢£§◊$®©®\'"†°`´¨≠∞±≤≥ªº¿¡√«»""''÷

These restrictions are not applicable to terminal symbols defined between single quotes in a grammar.

 Terminal symbols may be mapped to one another through one or several (non-erasing) .i.**homomorphism**;**s**. A homomorphism named 'OCT' may for instance be used to define octaves in a terminal alphabet of simple notes in the English convention:

OCT

C0 --> C1 --> C2 --> C3 --> C4 --> C5 --> C6 --> C7 --> C8 --> C9 --> C10

C#0 --> C#1 --> C#2 --> C#3 --> C#4 --> C#5 --> C#6 --> C#7 --> C#8 --> C#9 --> C#10

Db0 --> Db1 --> Db2 --> Db3 --> Db4 --> Db5 --> Db6 --> Db7 --> Db8 --> Db9 --> Db10

D0 --> D1 --> D2 --> D3 --> D4 --> D5 --> D6 --> D7 --> D8 --> D9 --> D10

D#0 --> D#1 --> D#2 --> D#3 --> D#4 --> D#5 --> D#6 --> D#7 --> D#8 --> D#9 --> D#10

etc...

Similarly, a homomorphism named 'TRANS' would for instance transpose all simple notes one semitone higher:

TRANS

C0 --> C#0 --> D0 --> D#0 --> ...

These homomorphisms are edited in the .i."Alphabet" window; and stored in ".i.-ho.<name>;" files. See for instance "-gr.MyMelody", §4.10 infra. Examples of homomorphisms doing .i.tonal transformations; other than transpositions may be found in ".i.-gr.Ruwet;" and in ".i.-gr.cloches1;".

A .i.**variable**; is a symbol bound to be rewritten as a string of terminals and/or variables (in a **derivation** of the grammar). The label of a variable is either an alphanumeric string starting with an uppercase character, or written between |vertical bars|, and may otherwise contain any character in the set:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz\-\_#@\*%$"%'•

When BP2 reads a grammar it first assumes any unknown word to be a variable. Therefore, if that word is not between vertical bars and does not begin with an uppercase character, an error message is returned. Be careful to separate variables with spaces or tabulations. If you write "XYZ", a variable named "XYZ" is created although you perhaps meant "X", "Y" and "Z"... However, spacing is not necessary for stringing together predefined tokens like terminal symbols.

The concepts of .i.terminal symbol;s and .i.variable;s are slightly different in BP grammars and conventional formal grammars. In formal grammars, "terminals" are the symbols that cannot be rewritten; in BP grammars they denote labels of hypothetic sound-objects. BP2 makes no difference between terminal symbols and variables as far as derivations are concerned. This means that even BP terminal symbols might be used as variables in production rules. (Not recommended)

**4.2 Starting symbol**

A rewriting system operates transformations on strings of symbols. In formal grammars, a special symbol is used to indicate the initial work string (the .i.**starting symbol**;). This initial symbol is generally notated 'S'. It is also the default one assumed by BP2. You may nevertheless wish to use a different symbol, or even a string of variables as a .i.start string;. See QuickStart §3.9.

**4.3 Patterns in BP grammars**

A .i.**pattern**; is a string of terminals and variables derived in a particular way: the derivation is obtained by replacing every variable with another pattern. For instance, if {a,b,c,...z} is a terminal alphabet, pattern "XfYXXY" may be derived as "abfYababY", "XfaZccXXaZcc", "abfaZccababaZcc", "abfaedccababaedcc", etc. using the following derivations:

X => ab , Y => aZcc , Z => ed

The set of all possible **terminal derivations** (e.g. "abfaedccababaedcc") of a pattern is called the **language** generated by this pattern. A .i.**pattern language**; is infinite unless some rules are used to define a limited number of acceptable derivations.

A grammar containing patterns may be called a .i.**pattern grammar**;. See for instance §4.5.

**4.4 BP2 grammars**

The language generated by a BP2 grammar is the intersection of a pattern language and an unrestricted (phrase-structure or type 0) language (Salomaa 1973:15, Révész 1985:6, Bel 1989). A .i.**production rule**; is written

P --> Q

in which "P" (the .i.**left argument of a rule**;) and "Q" (the .i.**right argument of a rule**;) are strings of variables and/or terminals; the left argument should contain at least one variable.

The right argument may be the .i.**empty string**;. In formal language literature the empty string is often notated 'λ', for instance:

S --> λ

Since 'λ' is not accessible on the keyboard you may type any of the following:

S -->

S --> lambda

S --> nil

S --> empty

S --> null

Reserved tokens 'lambda"', 'nil', 'empty' and 'null' may not be redefined as terminal symbols.

In derivations of a formal grammar, a string containing several occurrences of a single variable is normally not handled as a pattern: each occurrence may be derived independently. Therefore a special format is needed for patterns. If for instance "YabZXcYXdX" is a pattern, it is necessary to indicate explicitly that all occurrences of "X" (resp. "Y") must be derived identically. The notation used in BP2 grammars is

(=Y) abZ (=X) c (:Y) (:X) d (:X)

in which brackets flagged with '=' are .i.**reference expression**;**s** and brackets with ':' .i.**slave expressions**;. Multilevel .i.bracketing; is possible. For instance, the following BP2 grammar

S --> (= (= X) S (:X)) (: (= X) S (:X))

S --> nil

X --> a

X --> a S b

X --> b

generates .i.self-similar fractal;s.

Bracketing is also used to indicate .i.homomorphic transformation;s. For instance, given the TRANS and OCT homomorphisms (see §4.1 supra), pattern

TRANS TRANS (=X) (:X) OCT TRANS TRANS TRANS TRANS (:X)

would represent three occurrences of the same structure of simple notes "X", the first one being two semitones higher, and the last one four semitones and one octave higher than the second one.

Brackets and other structural markers are removed once a terminal string (a **sentence** or **item**) has been generated.

BP grammars are generally arranged in several layers of .i.**subgrammar**;**s** separated with lines of hyphens. These may be called .i.**transformational grammar**;**s** (in a non-linguistic sense, see Kain 1981:24-5). Each **subgrammar** may have its own .i.**derivation mode** (grammar);:

-- ".i.RND;" mode: rules are applied in a random order. (Default mode)

-- ".i.LIN;" mode: candidate rule chosen randomly among rules yielding a leftmost derivation.

-- ".i.ORD;" mode: candidate rules are applied in the order in which they appear in the grammar.

-- ".i.SUB;" mode: same as "RND", but all occurrences of the selected rule's left argument in the work string are rewritten simultaneously. In this mode, for instance, **p-substitutions** and other kinds of automatic (infinite) sequences may be generated. (See §4.14 infra.) These derivations stop either when a predefined buffer length has been reached, or the weights of all candidate rules are null as the result of having been dynamically decremented during computation, or by clicking the mouse.

-- ".i.SUB1;" mode: same as "SUB", but only one rewriting of variables will be allowed. This mode is much faster than "SUB" and may be used when it is certain that the first substitution yields only terminal symbols. (See example in ".i.-gr.Mozart;", §5.4 of QuickStart)

Below is an illustration of a "LIN" subgrammar (".i.-gr.tryLIN;"). We need to produce strings of 'a', 'b', 'c' with lengths 18, in which no two consecutive

occurrences of b are found, and c's always come in pairs. The following grammar does it:

ORD

S --> X X X X X X X X X X X X X X X X X X

----------------------------------------------------------

LIN

X --> a

#b X --> #b b [Negative context. Never two consecutive 'b']

X ? --> c Y ? [Can't be applied if "X" is the rightmost symbol]

Y X --> c [This rule is length-decreasing]

Typical productions of this grammar are:

b a a b c c a c c a b a a b a b c c

b c c b c c c c c c a b a b c c b a

b c c c c c c c c c c c c c c b a a

a c c b a c c a a c c c c b c c c c

c c a a c c a b a c c a a b c c b a

a b a b a b a c c b a b a a b c c a

b c c a b a a a c c a b a c c c c b

a c c c c c c b c c c c a a a c c b

a a a c c a b c c a c c a b c c c c

The detailed derivation of the first example is self-explanatory:

S

X X X X X X X X X X X X X X X X X X

b X X X X X X X X X X X X X X X X X

b a X X X X X X X X X X X X X X X X

b a a X X X X X X X X X X X X X X X

b a a b X X X X X X X X X X X X X X

b a a b c Y X X X X X X X X X X X X X

b a a b c c X X X X X X X X X X X X

b a a b c c a X X X X X X X X X X X

b a a b c c a c Y X X X X X X X X X X

b a a b c c a c c X X X X X X X X X

b a a b c c a c c a X X X X X X X X

b a a b c c a c c a b X X X X X X X

b a a b c c a c c a b a X X X X X X

b a a b c c a c c a b a a X X X X X

b a a b c c a c c a b a a b X X X X

b a a b c c a c c a b a a b a X X X

b a a b c c a c c a b a a b a b X X

b a a b c c a c c a b a a b a b c Y X

b a a b c c a c c a b a a b a b c c

A "RND" subgrammar instead of a "LIN" subgrammar would yield, for instance

S

X X X X X X X X X X X X X X X X X X

X X X b X X X X X X X X X X X X X X

X X X b X X X X X a X X X X X X X X

X X X b X X X X X a X X c Y X X X X X

X X a b X X X X X a X X c Y X X X X X

X X a b X X X a X a X X c Y X X X X X

X X a b X X b a X a X X c Y X X X X X

X X a b X X b a X a X X c c X X X X

X X a b X b b a X a X X c c X X X X

X X a b X b b a X a X X c c X X b X

X X a b X b b a X a b X c c X X b X

X X a b X b b a X a b X c c X c Y b X

X X a b X b b a X a b X c c c Y c Y b X

X X a b X b b a b a b X c c c Y c Y b X

b X a b X b b a b a b X c c c Y c Y b X

b X a b X b b a b a b a c c c Y c Y b X

b a a b X b b a b a b a c c c Y c Y b X

b a a b a b b a b a b a c c c Y c Y b X

b a a b a b b a b a b a c c c Y c Y b a

b a a b a b b a b a b a c c c Y c Y b a

which (like many context-sensitive grammars) does not produce a terminal derivation.

Each rule has its own .i.derivation mode (rule);. As suggested above, in ".i.LIN;" grammars the leftmost occurrence of the first argument of the rule is searched for. However, in ".i.RND;" or ".i.ORD;" grammars the derivation mode needs to be specified. The default derivation mode, notated "RND", means that the derivation position will be taken randomly among candidate positions in the work string. If you want to rewrite the leftmost (resp. rightmost) occurrence of the left argument of the rule, insert ".i.LEFT;" (resp. ".i.RIGHT;") before the first argument of the rule (after its weight). Note that "LEFT" or "RIGHT" rules take significantly less computation time.

You must also keep in mind that a "RND" grammar with "LEFT" rules does necessarily yield a leftmost derivation. For instance, the following variant of the ".i.-gr.tryLIN;" grammar

ORD

S --> X X X X X X X X X X X X X X X X X X

----------------------------------------------------------

RND

LEFT X --> a

LEFT #b X --> #b b

LEFT X ? --> c Y ?

LEFT Y X --> c

produces the (incorrect) derivation:

S

X X X X X X X X X X X X X X X X X X

b X X X X X X X X X X X X X X X X X

b a X X X X X X X X X X X X X X X X

b a a X X X X X X X X X X X X X X X

b a a b X X X X X X X X X X X X X X

b a a b c Y X X X X X X X X X X X X X

b a a b c Y a X X X X X X X X X X X X [Here it failed because "X --> a" was also candidate]

b a a b c Y a c Y X X X X X X X X X X X

b a a b c Y a c Y a X X X X X X X X X X

b a a b c Y a c Y a b X X X X X X X X X

b a a b c Y a c Y a b a X X X X X X X X

b a a b c Y a c Y a b a a X X X X X X X

b a a b c Y a c Y a b a a b X X X X X X

b a a b c Y a c Y a b a a b a X X X X X

b a a b c Y a c Y a b a a b a b X X X X

b a a b c Y a c Y a b a a b a b c Y X X X

b a a b c Y a c Y a b a a b a b c Y c Y X X

b a a b c Y a c Y a b a a b a b c Y c c X

b a a b c Y a c Y a b a a b a b c Y c c b

b a a b c Y a c Y a b a a b a b c Y c c b

A partial control of derivations is possible either after an interruption or before starting a production, by choosing the "Step-by-step" and "Select candidates" options. (See QuickStart §12)

**.i.4.5 Destroying structure;s**

Consider the three-level .i.**pattern grammar**; (see ".i.-gr.tryDESTRU;"):

RND

<2-1> S --> (= (= X) S (:X)) (: (= X) S (:X))

<2-1> X --> Y

<2-1> X --> Y S Z

<2-1> X --> Z

----------------------------------------

ORD

LEFT S --> lambda

----------------------------------------

LIN

[A line is missing here...]

X X --> abca

Z Y --> abc

Y Z --> cba

Y Y --> cbbc

Z Z --> lambda

This grammar would not produce strings of terminal symbols because the first subgrammar produces strings of X's and Y's mixed with structural symbols "(=", "(:", ")". These symbols should be deleted before subgrammar #3 is applied. Procedure ".i.\_destru;" takes care of it. This instruction should be placed on top of the third subgrammar:

LIN

\_destru

X X --> abca

Z Y --> abc

...

With the "\_destru" procedure, terminal strings on the alphabet {a,b,c} are generated, for instance:

cbbcabcabcabcabccbbcabcabcabcabc

abcabcabcabcabcabcabcabc

cbacbbccbbccbacbacbbccbbccba

cbacbacbacba

cbaabccbaabc

cbbccbbccbbccbbccbbccbbc

abcabcabcabcabcabcabcabc

**4.6 Weights**

A .i.**weight**s in grammar; in range [0, 32767] may be attached to each rule. They are used when several rules are candidates in "RND", "LIN", "SUB" or "SUB1" grammars. A rule with weight <0> is .i.**inactive** rule;. The weight value is indicated between angle brackets, e.g. <43>. Default weight is <127>. It is convenient to use weights in range 0..127 so that they may be set by MIDI controllers, using parameters K1, K2... (see QuickStart §6).

Weights may be **dynamically controlled**.i.dynamic weight;: any rule may gain a positive or negative weight increment each time it is applied. For instance, a rule with weight <100-30> will have successive weights 100, 70, 40, 10 and 0, and therefore cannot be applied more than four times in the same production.

A good idea for checking a grammar is to set the weights of some rules to <1-1> (or <n-n> for any integer *n*). These rules can be used only one time. Once an item has been produced, produce a new one without resetting rule weights (the option is available by typing cmd-option space): the next item will be based on different rules since former ones have become inactive.

When using a subgrammar with dynamically decrementing weights, derivations of the work string stop when all candidate rules have weight <0>. This is a common way of avoiding endless derivations.

Changing weights dynamically on context-sensitive rules is a simple way to let patterns "emerge" randomly: some rules are becoming more active while they "inhibit" others.

Weights may be **infinite**. .i.infinite weight;This is necessary when a rule should absolutely be applied as soon as it becomes a candidate. (BP2 actually selects the *first* candidate rule with an .i.infinite weight;) An infinite weight is notated "<∞>". On US keyboards, character '∞' is obtained by typing '5' with the 'option' key down.

**4.7 Metavariables**

.i.metavariable;Wild cards notated '?1', '?2', etc. may be used in production rules. A .i.wild card; may be replaced with any terminal symbol, simple note, variable, bracket, out-time object, synchronisation tag, time-pattern or performance parameter. For example, a rule like

?1 ?2 ?3 ?1 --> (= ?1) (= ?2) ?3 \* (: ?2) (: ?1)

rewrites "X Y Z X" as "(= X) (= Y) Z \* (: Y) (: X)". (Symbol '\*' is an homomorphism.)

'?' is an old notation that may be replaced with '?n' for any value of n.

**4.8 Tempo and gaps**

If no indication is given, a sequence of symbols is interpreted as "one symbol per time unit". We mean **symbolic** time units counted on a .i.**symbolic tempo**;. Sound-object sequence "a b c d e f" may also be notated "/1 a b c d", where "/1" is called an .i.**explicit tempo marker**;. Similarly, if the same sequence is to be interpreted three times faster it is notated "/3 a b c d e f".

Using explicit tempo markers makes it possible to change tempo within the string. For instance, in sequence

/2 a b c d e f /3 g h i j k l m n o

'a' ... 'f' are interpreted at speed 2 (two sound-objects per beat) while 'g' to 'o' are interpreted at speed 3. (This may also be viewed as a tempo acceleration of 3/2.) If '\_' (empty sound-object) is used to denote a prolongation of the preceding object, the same expression may equivalently be notated:

/6 a\_ \_ b\_ \_ c\_ \_ d\_ \_ e\_ \_ f\_ \_ g\_ h\_ i\_ j\_ k\_ l\_ m\_ n\_ o\_

Gaps (.i.silence;s or rests) are written as hyphens or integer numbers. The following notations are equivalent:

/2 a b - - c d /3 e - - - - f g h

/2 a b 2 c d /3 e 4 f g h

Rational numbers may also be used to indicate .i.**fractional gap;s**, e.g.

/1 a b /2 c d e f 4/3 g h

in which 'a' and 'b' are interpreted at speed one, 'c', 'd', 'e', 'f', "g" and "h" at speed two while sequences "cdef" and "gh" are separated with a silence of duration 4/3. Since the "4/3" silence occurs at speed two its actual .i.symbolic duration; is 4/3 x 1/2 = 2/3. Here again BP2 will expand the representation to

/6 a \_ \_ \_ \_ \_ b \_ \_ \_ \_ \_ c \_ \_ d \_ \_ e \_ \_ f \_ \_ - - - - g \_ \_ h \_ \_

where the "4/3" gap is represented as "- - - -" (or "- \_ \_ \_" equivalently). The following representations are equivalent:

/6 a \_ \_ \_ \_ \_ b \_ \_ \_ \_ \_ c \_ \_ d \_ \_ e \_ \_ f \_ \_ - \_ \_ \_ g \_ \_ h \_ \_

/6 a \_ \_ \_ \_ \_ b \_ \_ \_ \_ \_ c \_ \_ d \_ \_ e \_ \_ f \_ \_ 4 g \_ \_ h \_ \_

**4.9 Object concatenation**

Normally a string "abcb" represents a sequence of objects 'a','b','c' such that:

'b' starts at the moment 'a' stops; 'c' starts at the moment 'b' stops; 'b' starts again at the moment 'c' stops.

What if we want 'b' to continue while 'c' is on? In other words, we wish to represent:



**Fig.5 Using the concatenation symbol**

This can be done using the .i.**concatenation symbol**; '&'. The proper representation is

a b& c &b

Beware that spaces are meaningful in this representation: you can't write "b&c". More complex structures can be defined in a similar way. For instance,

a b& c& d c &c b &b

may be interpreted as:



**Fig.6 Using the concatenation symbol (more complex)**

.i.Object concatenation; is used for representing objects which belong to several substructures, i.e. that are not structured as a tree hierarchy. See for instance the grammar generating Ames' example in §4.10.

**.i.4.10 Polymetric expression;s**

.i.Polymetric expression;s are useful for string representations of .i.**polyphonic music**;. A simple .i.polymetric expression; is shown figure 7 in staff notation, BP2 graphics and *phase diagram*. The latter is a table containing pointers to the instances of sound-objects 'C4', 'E#3', etc.



**Fig.7 Staff notation,** .i.**phase diagram**; **and BP2 graphic display of a** .i.**polymetric expression**; **notated "{1, C4 -, - E#3 G3, A#5, - D5}"**

This expression is notated "{1, C4 -, - E#3 G3, A#5, - D5}" on a BP2 text score. It shows five sequences (*fields*) separated by commas. The leftmost field contains '1', the *symbolic duration* of the expression. With a metronome set to 45 beats per minute, the resulting physical duration is 1.33 seconds.

The *polymetric expansion algorithm* imbedded in Bol Processor coerces all sequences of the structure to the same symbolic duration. The second field contains a note 'C4' followed by a silence '-', both of which will be treated as quavers. The third field contains the sequence "- E#3 G3" which is performed as a triplet.

Thus, the .i.polymetric expression; "{1, C4 -, - E#3 G3, A#5, - D5}" is expanded to "/6 {C4\_ \_ -\_ \_,-\_ E#3\_ G3\_,A#5\_ \_ \_ \_ \_,-\_ \_ D5\_ \_}" as suggested by the phase diagram. Symbol '\_' is a prolongation of the preceding sound-object, and '/6' indicates a change of tempo after which durations are divided by 6.

The phrase shown figure 7 is an excerpt of a musical example imitating Steve Reich's style. A grammar generating examples in this style, composed by Thierry Montaudon, is shown fig.8.

S --> \_velcont \_vel(50) Part1 Part2 Part3

Part1 --> A A A A A A A A

Part2 --> B B B' B' C C D D D D E E E E

Part3 --> C C C D E E E E C D C D E E E E C C C B' B' B A A A A A

A --> {1, C4 -,\_vel(40) - E#3 G3, A#5, - D5}

B --> {A A, \_vel(60) C2}

B' --> {B, - F5}

C --> {B, \_vel(55) - C5}

D --> {B, - C4 F5 E#4}

E --> {D, D#4 F4 C5 G#3}

**Fig.8 A grammar producing a piece *à la* Steve Reich**

In this grammar, velocities are set by controls "\_vel(x)" and are interpolated throughout the piece due to instruction "\_velcont". The upper four rules define the deep structure of the piece. The lower ones use variables 'A', 'B', 'C'... to construct polymetric structures stringed together in each part of the piece.

Consider another example. Let {a b , c d e} be a sound structure in which two sound-object sequences: "a b" and "c d e" are superimposed. The interpretation chosen by BP2 is

{a \_ \_ b \_ \_ , c \_ d \_ e \_}

in which '\_' is a prolongation of the preceding sound-object. The two sequences may be written on a two-line .i.phase diagram



which exclusively states that *'a' starts with 'c', 'd' starts before 'b' which in turn starts before 'e'...*

The interpretation of this .i.polymetric expression; automatically sets the tempo of "cde" at a speed which is 3/2 that of "ab". This property may be used for indicating any fractional relative change of tempo. For example,

/1 {4, a b c - - } {2, d e f} {5/3, g h}

indicates that sequence "a b c - -" must be adjusted to fit exactly within 4 beats, then "d e f" within 2 beats, then "g h" within 5/3 beats. BP2 performs the necessary calculations and stretches the final representation. The fact that a representation is ‘stretched’ does not mean it will be played slower. BP2 calculates a time-scale factor to adjust its duration accordingly.

Integers, fractions or hyphens indicating durations must appear in the **first** field of a .i.polymetric expression;, because if an expression contains no explicit tempo marker its default duration is that of the first field. Compare for instance

/1 a b {c d, e f g} h i = /3 a \_ \_ b \_ \_ {c \_ \_ d \_ \_ , e \_ f \_ g \_ } h \_ \_ i \_ \_

with:

/1 a b {e f g , c d} h i = /2 a \_ b \_ {e \_ f \_ g \_ , c \_ \_ d \_ \_ } h \_ i \_

Another feature useful for dealing with polymetric structures is the handling of .i.**undetermined rest**;**s**: notated '\_rest' or '...' (Use option semi-colon on a US keyboard, never type three periods) Each field in a .i.polymetric expression; may contain one undetermined rest appearing anywhere in the sequence. BP2 first determines the structure ignoring fields with undetermined rests, then it scans again fields with undetermined rests. Some among them have fixed durations because they contain an explicit tempo marker, e.g.:

{/1 fa5 {la5 si5,do6 re6 mi6}, do1 \_rest /2 fa1 }

in which the second field contains "/2", i.e. two sound-objects per time unit. It also means that "do1" should be performed at (default) one-sound-object-per-time-unit tempo, which is formally notated:

{/1 fa5 {la5 si5,do6 re6 mi6}, /1 do1 \_rest /2 fa1 }

The symbolic duration of the first field (hence, of the second field as well) is that of sequence "/1 fa5 la5 si5", i.e. three units. The duration of the second field is 3/2 units plus the undetermined rest. Therefore the undetermined rest is also 3/2 units, which could have been written:

{/1 fa5 {la5 si5,do6 re6 mi6}, do1 3/2 /2 fa1 }

An error message is generated if there is insufficient time for a rest. An undetermined rest is meant to yield the *simplest* possible expression. For example,

{fa5 {la5 si5,do6 re6 mi6}, do1 re1 mi1 ... fa1 }

= {fa5 {la5 si5,do6 re6 mi6}, do1 re1 mi1 - - fa1 }

= {fa5 \_ {la5 \_ si5 \_ , do6 \_ re6 \_ mi6 \_ }, do1 re1 mi1 - - fa1 }

(See Bel 1990a, 1991, 1992).

Let us now compare .i.polymetric expression;s with .i.**event table**;**s** used in MIDI sequencers. The following musical fragment is borrowed from the *COMPOSE Tutorial and Cookbook* (Ames 1989:2):



**Fig.9 Score display**



**Fig.10 Piano-roll display**

The list of events for this example is given in the .i.**event table**; (ibid:3):

|  |  |  |
| --- | --- | --- |
| Period.667.667.667.51.2502.25 | Duration.6671.333.667.53.52.25.25 | PitchR *[rest]*F#3F5:A5G#3:E5:G5Bb4C6:E6G#5:B6 |

where "period" stands for the time elapsed from the on-setting of a note (or rest or chord) to the on-setting of its successor.

A possible (yet arguable) structural analysis of this example is



**Fig.11 A structural analysis**

The score shown Fig.11 may be generated by the following grammar. Variables A1, A2, etc. must be written between vertical bars so that they are not confused with terminal symbols (musical notes A1, A2, etc.).

S --> {|A|,... |B|}

A --> {2, |A1|,2 |A2|} {4, |A3|}

|A1| --> {|A11| ..., - |A12|}

|A3| --> |A31| |A32|

|A11| --> -

|A12| --> {2,F#3}

|A2| --> {F5,A5}

|A31| --> {1/2,G#3,E5,G5}

|A32| --> {3/2,Bb4&} {2,&Bb4}

B --> {1/4,G#5&,C6,E6,B6&} {2,&G#5,&B6}

... (other rules in the same grammar)

yielding a unique derivation:

 {{2 ,{- ...,-{2 ,F#3}},2 {F5,A5}}{4 ,{1 /2 ,G#3,E5,G5}{3 /2 ,Bb4&}{2 ,&Bb4}},...{1 /4 ,G#5&,C6,E6,B6&}{2 ,&G#5,&B6}}

Time information is redundant in this representation (evidently, 4 = 1/2 + 3/2 + 2), but it is consistent. BP2 will produce the following expanded representation

 /12{{-\_ \_ \_ \_ \_ \_ \_ F#3\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ ,-\_ \_ \_ \_ \_ \_ \_ -\_ \_ \_ \_ \_ \_ \_ {F5\_ \_ \_ \_ \_ \_ \_ ,A5\_ \_ \_ \_ \_ \_ \_ }}{G#3\_ \_ \_ \_ \_ ,E5\_ \_ \_ \_ \_ ,G5\_ \_ \_ \_ \_ }Bb4\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ & &Bb4\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ ,-\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ -\_ \_ {G#5\_ \_ &,C6\_ \_ ,E6\_ \_ ,B6\_ \_ &}{&G#5\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ ,&B6\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ }}

which is internally represented:

 /12 {/12 {/6 /18 - /9 F#3 /18 /6 ,/18 - -{/18 F5,/18 A5}/18 }/12 /12 {/24 G#3,/24 E5,/24 G5}/12 /8 Bb4&/12 /6 &Bb4 /12 /12 ,/12 /48 - - - - - - - - - - - - - - - /12 {/48 G#5&,/48 C6,/48 E6,/48 B6&}/12 {/6 &G#5,/6 &B6}/12 }/12

The philosophy behind .i.polymetric representation; is that it should contain a minimum amount of time information, for two major reasons:

(1) there is a lot of time information that the machine is able to calculate, for instance here the duration of chord {F5, A5};

(2) timing information should be based on final items, therefore on contextual information regarding tempo.

**.i.4.11 Period notation**

In the same way it deals with superimposed sequences, the polymetric expansion algorithm works out equal symbolic durations between beat separators notated '•' -- the "period notation." A note sequence in period notation and the context-free grammar it originated from (composed by Harm Visser) are shown Fig.12. See ".i.-gr.acceleration;" in the VISSER & MONTAUDON folder.

In this example, beats contain increasing numbers of notes resulting in an accelerating movement. Velocity assignments have of course no effect on durations.

S --> \_vel(60) A B \_vel(65) C D \_vel(70) E F \_vel(75) G \_vel(77) H \_vel(80) I \_vel(85) J \_vel(87) K \_vel(90) L

A --> E2 •

B --> D2 A

C --> B2 B

D --> G2 C

E --> F#2 D

F --> A#2 E

G --> C2 F

H --> G#2 G

I --> A2 H

J --> D#2 I

K --> C#2 J

L --> F2 K

BP2 score: *Velocity controls have been left out*

E2 • D2 E2 • B2 D2 E2 • G2 B2 D2 E2 • F#2 G2 B2 D2 E2 • A#2 F#2 G2 B2 D2 E2 • C2 A#2 F#2 G2 B2 D2 E2 • G#2 C2 A#2 F#2 G2 B2 D2 E2 • A2 G#2 C2 A#2 F#2 G2 B2 D2 E2 • D#2 A2 G#2 C2 A#2 F#2 G2 B2 D2 E2 • C#2 D#2 A2 G#2 C2 A#2 F#2 G2 B2 D2 E2 • F2 C#2 D#2 A2 G#2 C2 A#2 F#2 G2 B2 D2 E2 •

**Fig. 12 A grammar producing an accelerating sequence of notes, and the resulting item in BP2 score notation**

Once the grammar has been typed, the user may select "Produce items" to get a text or graphic display of its production, listen to it on the MIDI output, and optionally produce a Csound score. The output of the "acceleration grammar" of figure 2 is displayed on a piano-roll score:



**Fig.13 A piano-roll of the item produced by the grammar of Fig.12**

**4.12 Remote context**

Standard formal (Chomsky) grammars make it very difficult (although theoretically possible) to control productions on the basis of a .i.**remote context**;, i.e. the occurrence of a string located *anywhere* to the left or right side of the derivation position. Therefore a special syntax of remote contexts is available in BP2.

Remote contexts are represented between ordinary brackets in the .i.left argument of a rule;. These brackets are not confused with pattern delimiters because they neither contain '=' nor ':'.

For instance, a rule like

(a b c) X Y (c d) --> X e f

means that "X Y" may be rewritten as "X e f" only if "a b c" is found somewhere before "X Y" in the string under derivation, and "c d" somewhere after "X Y". Note that "X" itself is a left context in the sense of conventional generative grammars.

A .i.remote context; may contain any string in BP syntax, including string patterns and metavariables. It may also be negative. For instance,

#(a b c) X --> c d e

means that "X" may be rewritten as "c d e" only if not preceded by "a b c" in the string under derivation.

Below is a typical grammar using remote contexts (see ".i.-gr.bells;"). The problem was to generate sequences of permutations of 'a', 'b', 'c', 'd', using simple transformations, given that sequences are never repeated. (This problem is borrowed from the traditional art of composing bell-ringing tunes, see Bel 1990c,1992) The sequence must begin and end with "ABCD", hence the infinite-weight rule that terminates the generation.

RND

S --> /4 Tune

Tune --> Canonic New B A C D X12 End

Tune --> Canonic New A C B D X23 End

Tune --> Canonic New A B D C X34 End

Tune --> Canonic New B A D C X1234 End

----------------------------------------------------------

LIN

<∞> A B C D ?1 End --> A B C D [Infinite weight]

#(?1 ?2 ?4 ?3) ?1 ?2 ?3 ?4 X12 --> ?1 ?2 ?3 ?4 New ?1 ?2 ?4 ?3 X34

#(?2 ?1 ?4 ?3) ?1 ?2 ?3 ?4 X12 --> ?1 ?2 ?3 ?4 New ?2 ?1 ?4 ?3 X1234

#(?2 ?1 ?4 ?3) ?1 ?2 ?3 ?4 X34 --> ?1 ?2 ?3 ?4 New ?2 ?1 ?4 ?3 X1234

#(?2 ?1 ?3 ?4) ?1 ?2 ?3 ?4 X34 --> ?1 ?2 ?3 ?4 New ?2 ?1 ?3 ?4 X12

#(?2 ?1 ?4 ?3) ?1 ?2 ?3 ?4 X23 --> ?1 ?2 ?3 ?4 New ?2 ?1 ?4 ?3 X1234

#(?2 ?1 ?3 ?4) ?1 ?2 ?3 ?4 X1234 --> ?1 ?2 ?3 ?4 New ?2 ?1 ?3 ?4 X12

#(?1 ?3 ?2 ?4) ?1 ?2 ?3 ?4 X1234 --> ?1 ?2 ?3 ?4 New ?1 ?3 ?2 ?4 X23

#(?1 ?2 ?4 ?3) ?1 ?2 ?3 ?4 X1234 --> ?1 ?2 ?3 ?4 New ?1 ?2 ?4 ?3 X34

-----------------------------------------------------------

ORD

[These rules are needed in case the infinite-weight rule could not be applied]

LEFT X12 --> lambda

LEFT X34 --> lambda

LEFT X23 --> lambda

LEFT X1234 --> lambda

LEFT End --> lambda

LEFT New --> lambda

LEFT Canonic --> A B C D

-----------------------------------------------------------

SUB

[Tuning bells... See substitutions §4.14]

A B --> do3 B

A #B --> do4 #B

B --> sol3

B --> sol4

C --> re4

C --> re5

D A --> mi4 A

D #A --> mi5 #A

**4.13 Metagrammars**

All symbols used for representing a grammar may be the terminals of a .i.**metagrammar**; whose role is to generate a set of (related) grammars. For instance, the following metagrammar

RND [A grammar is a set of rules]

<1> S --> 'RND'; Ri

<10> S --> S ; R

----------------------

RND [Define context-free rules in Chomsky normal form]

Ri --> 'S' '-->' Arg2

R --> Arg1 '-->' Arg2

Arg1 --> Variable

<1> Arg2 --> Constant

<5> Arg2 --> Variable Variable

------------------------

RND

Variable --> X

Variable --> Y

Variable --> Z

Constant --> a

Constant --> b

Constant --> c

Constant --> d

generates any λ-free context-free grammar with variables {X,Y,Z} and terminal symbols {a,b,c,d}. 'S', 'RND' and '-->' are terminal symbols of the metagrammar itself, hence the single quotes. Semicolons generated by this grammar are automatically converted to line feeds.

Most grammars generated by ".i.-gr.gramgene1;" produce empty languages. There are better ways of producing grammars. A grammar called ".i.-gr.gramgene2;" on the BP2 disk may be used to generate (non-empty-language) pattern grammars such as for instance:

-ho.abc

RND

S --> Y Z

Y --> X X

Z --> X X

Y --> b

Z --> d

Y --> c

Z --> (=(= Z )(: Z ))(:(= Z )(: Z )) Y

Y --> d

Z --> X X

X --> b

X --> X Z

X --> a

X --> (= X )(: X ) Y

**4.14 Substitutions**

".i.SUB;" grammars perform simultaneous rewriting of symbols in the work string. For instance,

SUB

S --> A B B A B B A

A B --> a B

B A --> B b

A A --> c c

A B A --> A d A

B B A --> B e A

B B --> f B

produces string "afeafeb" in one single ".i.**parallel derivation**;".

The string "ABBABBA" is rewritten "afeafeb" as shown Fig.14.



**Fig.14 A substitution in a "SUB" grammar**

.i.Substitution;s may be used for designing a general category of unidimensional .i.cellular automata; and some multidimensional ones.

If substitutions are performed on strings of terminal symbols, each step may produce an interesting sound-object sequence. For this reason, it is possible to instruct BP2 to .i.play all substitutions; in the "Improvize" mode. The option is given in the "Settings" dialog. See an application in ".i.-gr.koto3;".

**5. The time setting of sound-objects**

**5.1 The time setting problem**

Informally, .i.**instanciating** a sound-object; means dispatching to the sound processor all the messages that are defined in its prototype.

A naive interpretation of sequences of sound-objects would be to arrange all corresponding time intervals in a strictly sequential way. Duthen and Stroppa [1990] have suggested a more abstract approach, starting from the assumption that any sound-object may possess one or several time points playing a particular role, e.g. a climax. These points are called .i.**time pivot**;**s**. Further they suggest to construct sound structures using a set of synchronisation rules. Their approach is attractive but it is difficult to implement if the formalism of synchronisation rules remains too general. Therefore we retained a simplified version of Stroppa's idea, assigning each object one single pivot.

Let us for instance consider a polymetric structure {S1,S2,S3} derived as

{a \_ b c d \_ e , a \_ f \_ g h \_ , j i \_ a \_ i \_ }

yielding the .i.**phase diagram**;:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | \_ | b | c | d | \_ | e | NIL |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | \_ | f | \_ | g | h | \_ | NIL |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| j | i | \_ | a | \_ | i | \_ | NIL |

The definition of each object contains the relative location of its pivot and metrical properties allowing the calculation of its dilation ratio (see §3.1 supra).

Fig.15 is a graphic representation of a possible **instance** of this polymetric structure.



**Fig.15 A structure of sound-objects**

The .i.structure of time; is for instance an irregular pulsation represented with the vertical lines (.i.**time streak**;**s**). Objects 'c', 'f', "g" and 'a' have overlapping time-span intervals between the third and fourth streaks.

Vertical arrows indicate .i.time pivot;s. As shown with object 'e', the pivot is not necessarily a time point within the .i.time-span interval; of the sound-object.

This graphic represents the **default positioning** of objects with their pivots located exactly on .i.**time streak**;**s**. Although it is reasonable that instances of 'c', 'f' and 'a' are overlapping between the third and fourth streaks since they belong to distinct sequences which are performed simultaneously, it may not be acceptable that 'f' overlaps 'g' in a single sequence S2; the same with 'd' and 'e' in sequence S1. For similar reasons, it may not be acceptable that the time-

span intervals of 'j' and 'i' are disjoint in sequence S3 while no silence is shown in the symbolic representation.

How could one deal with a constraint such as <<*the end of sound-object 'f' may not overlap another sound-object in the same sequence>>* ? If object 'g' is relocatable then it may be delayed (shifted to the right) until the constraint is satisfied. We call this a .i.**local drift**; of the object. However, the end of 'g' will now overlap the beginning of 'h'. Assume that this also is not acceptable and 'h' is not relocatable. One should therefore look for another solution, for example truncate the beginning of 'h'. If this and other solutions are not acceptable then one may try to shift 'f' to the left or to truncate its end. In the first case it might be necessary to shift or truncate 'a' as well.

So far we mentioned a .i.constraint propagation; within one single sequence. In the .i.time setting algorithm; the three sequences are considered in order S1, S2, S3. Suppose that the .i.default positioning of objects; in S1 satisfies all constraints and no solution has been found to avoid the overlapping of 'f' and 'g' in S2. Another option is to envisage a .i.**global drift**; to the right of all objects following 'f' in S2. The global drift is notated Δ on Fig.16. All .i.time streak;s following the third one are delayed (see dotted vertical lines).



**Fig.16 A structure using** .i.**global drift**

This solution is labelled ".i.**Break tempo**;" because its effect is similar to the the .i.*organum* ;in conventional music notation. Although .i.global drift; increases the delay between the third and fourth streaks, the .i.**physical duration** of sound object;s is not modified because their dilation ratios have been calculated beforehand.

Now the positioning of objects in S2 is acceptable but it might have become unacceptable in S1: there may be a property of 'b' or 'c' saying that their time-span intervals cannot be disjoint, so that 'c' could be shifted to the left, etc. Evidently, whenever a global drift is decided the algorithm must start again from the first sequence.

The process of locating -- i.e. "instanciating" -- sound-objects, as illustrated in this example, is the task of the .i.**time setting algorithm**; imbedded in BP2.

**5.2 How to proceed**

To take advantage of the .i.time setting algorithm; you first define sound-object prototypes with .i.topological properties of sound-object;and .i.metrical properties of sound-object;properties. For instance, you may want that the beginning of sound-object 'a' always coincides with the end of the preceding object in the sequence. In this case, you should set property .i.ContBeg; to "true". (See §3.2) When the tempo is slow, BP2 may not be able to fulfil this condition unless object 'a' or the one(s) preceding its occurrence(s) have property .i.Reloc; (see §3.1). In general, given a set of objects and their properties, the time setting of an arbitrary structure may not yield a solution. If so then some constraints should be released or the process should be abandoned. In many cases there is an infinite number of solutions, out of which a finite subset called .i.**canonic solution**;**s** may be extracted.

When you click button "Play Selection" (or type cmd-p), if button ".i.Display time setting;" is not selected in the "Settings" window, then BP2 will handle everything without notice: if there are solutions to the time setting problem it will take the first one it finds. If there are no solutions it will first attempt to release all .i.continuity constraint;s (ignoring all .i.ContBeg; and .i.ContEnd; properties), then it will release only .i.overlapping constraint;s (ignoring .i.OverBeg; and .i.OverEnd; properties). If still unsuccessful it will release both types of constraints, in which case there is always a solution.

Since the time setting could become slow in very large structures, if you have set a limit to computation time and that limit is exhausted, BP2 will suddenly release all constraints so that an approximate solution is obtained very quickly.

If ".i.Display time setting;" is on then BP2 offers you the option to display the timings of objects and perhaps to make decisions on how to solve constraints. You may also select ".i.Show graphics;" to display each sequence. If you decide to display all canonic solutions you will be able to decide which one is appropriate. If you select ".i.Step time setting;" you will be prompted to make decisions on every sound-object that needs to be modified to fulfil constraints. For instance, if the beginning of a sound-object is overlapped by the end of the preceding object you may be offered, (1) to delay ("shift") that object, (2) to truncate its beginning, (3) to shift the preceding object, and (4) to truncate the end of the preceding object.

When a sequence is completed it appears both in a numeric (in the "Trace" window) and graphic displays while you are prompted to accept or reject it. If you selected "all .i.canonic solution;s" and reject the currently displayed solution, BP2 will display all solutions in a loop until you finally accept one of them or abort the procedure.

**6. Time quantization**

.i.**Phase diagram;s** (see for instance QuickStart §4.2) might have unwieldingly many columns representing very short time delays (even less than the current time resolution). Setting a .i.**time quantization**; (an acceptable delay below which two events may be seen as simultaneous, i.e. belong to the same column) helps BP2 to simplify .i.phase diagram;s. In many cases this is compulsory for working with complex items in a limited memory space.

The following example demonstrates the quantization process. (Use ".i.-da.tryQuantize;") Consider the .i.polymetric expression

{do4 {re4 mi4, fa4 sol4 la4}, si4 do5}

which is interpreted as:

{/6 do4{/6 re4 mi4, /9 fa4 sol4 la4}, /4 si4 do5}

with the phase diagram (if quantization is OFF):



Metronome setting mm = 60 produces the following item:



**Fig.17 No time quantization**

The time delay between two consecutive .i.time streak;s is 1000 / 6 = 166 milliseconds. This is also the delay between the off-setting of "si4" and the on-setting of "sol4". Let us now use a quantization of 200 ms forcing the interpreter to ignore shorter delays. The new item is shown Fig.18.



**Fig.18 Same item with 200 ms** .i.**quantization**;

The delay between the off-setting of "si4" and the on-setting of "sol4" is now exactly 200 milliseconds, although it might have become 0 milliseconds in a different context. This amounts to rewriting the item as:

/3{do4\_ \_ {re4\_ \_ mi4\_ \_ ,fa4\_ sol4\_ la4\_},si4\_ \_ \_ do5\_ \_ \_ \_}

with the phase diagram:



Let us take another example. The item shown Fig.14 of QuickStart:

b b <<f>>{5 ,a c b,f - f}a <<chik>> b <<sync>>

was assigned the phase diagram:



Here, for instance, the symbolic duration of "b\_ \_ b\_ \_" is six time units. With a mm = 400 metronome setting, the time delay between two .i.time streak;s (i.e. two columns of the .i.phase diagram;) was 50 milliseconds. If the quantization is set to 60 milliseconds, the phase diagram becomes



which yields the graphic score shown Fig.19.



**Fig.19 Item with 60 ms quantization**

Durations of the first two occurrences of 'b' have become unequal although the total duration of "bb" remained two beats (i.e. 300 milliseconds). In this way, 'a', 'f' and "<<f>>" still have identical on-setting dates. Thus, quantization simplifies the .i.phase diagram; as much as possible while maintaining the dates of events within the given range -- here, dates will match exact dates by ±60 milliseconds. In doing so, durations may be adjusted within the acceptable range, but events that were synchronised (on "vertical lines") **will remain synchronised**.

Setting .i.quantization; to any value higher than 50 milliseconds yields the same result. Why? Since the symbolic duration of the first occurrence of 'b' has become one time unit, it cannot become smaller, otherwise sequentiality of the first two occurrences of 'b' would be damaged. **The quantization used by BP2 never affects** .i.**sequentiality of sound-objects**;. This is a very important feature because any pair of sound-objects may be used to set a process on and off, e.g. NoteOn/NoteOff.

The main advantage of BP2 quantization is an .i.optimisation of computational space; and time: it should be used systematically in projects

where polymetric structures are likely to yield oversized .i.**phase diagram;s**. In effect, a quantization equal to the time resolution does not change final timings although it is a safeguard against memory overflow.

**7. "True" Bol Processor grammars and parsing**

"True" BP grammars are transformational grammars allowing the production and .i.**parsing**; of strings representing musical sequences. The formalism of true grammars.i.true BP grammar; is the one implemented in .i.Bol Processor BP1; (Bel 1990b:33-66) for the study of drum improvisation, plus "remote contexts" (§4.12 supra).

A grammar successfully compiled by BP2 is a .i.**true BP grammar**; only if using **neither** of the following features:

• .i.Erasing rule;s;

• .i.Polymetric expression;s (§4.10) and associated features;

• .i.Substitution;s SUB and SUB1 (§4.14);

• .i.Programmed grammar;s (flags) (Quickstart §11);

• .i.Destroying structure;s (§4.5);

• .i.Production procedures; in rules (§8.1);

• .i.Dynamic weight assignment; (§4.6) or weights controlled interactively (QuickStart §6);

• .i.Metagrammar;s (§4.13);

When compiling a grammar, BP2 checks whether or not it is a true BP grammar. Nothing will happen at the time items are produced, but BP2 will refuse to analyse (parse) items if the current grammar is not "true BP".

A .i.true BP grammar; may be used to perform a .i.**membership test**; on arbitrary strings in order to check whether or not it might have been produced by the grammar. However, there are limitations for the validity of this test. These limitations are discussed in great detail in (Kippen & Bel 1992). (Also see Bel & Kippen 1992). Note for instance that the rewrite symbol "-->" is improper in BP grammars and should be replaced with "<-->" meaning that the rule is valid for both production and analysis. Thus, symbols
"-->" and "<--" indicate that a rule is used exclusively in production or analysis, respectively. Other restrictions of BP grammars deal with the order of rules in each subgrammar. The parsing algorithm takes rules in the opposite order of their occurrence in the subgrammar. Therefore, the last candidate rule in each subgrammar is the one that will first be used repeatedly (until it is no more candidate). As a consequence, its right argument should not be a substring of the .i.right argument of a rule; appearing higher in the subgrammar. (This is called the .i.**chunk rule**;, informally: *large patterns are recognised first.*)

To activate .i.parsing;, first generate templates ("Action" menu) and then select an item or a set of items in any text window. Select "Analyze" in the "Action" menu (or type cmd-j). BP2 will parse all items and return the resulting strings, whenever unsuccessful, in the "Trace" window.

If parsing was unsuccessful it may be important to see at what stage of the analysis an incorrect workstring was produced. Use "Step by step" and restart the analysis on this particular item. The "Repeat computation" button is inactive because parsing is deterministic.

**8. Procedures**

Formal grammars may be seen as declarative descriptions of string languages. A frequently raised question addresses the possibility of encapsulating .i.**procedural knowledge**; into .i.declarative knowledge;. In a formal grammar this may be accomplished by "flags" (see ".i.Programmed grammar;s", QuickStart §11) or by other procedures, see for instance ".i.Destroying structure;s" §4.5.

Here we deal with .i.**procedure;s** that may be inserted in subgrammar rules. Every procedure name starts with an underline character. Current procedures are

.i.\_destru; .i.\_goto; .i.\_failed; .i.\_repeat; .i.\_stop

that control derivations, and

.i.\_print; .i.\_printOn; .i.\_printOff; .i.\_stepOn; .i.\_stepOff; .i.\_traceOn; .i.\_traceOff

that modify the trace output, not the derivation itself.

Note that the names of procedures are not case-sensitive. Thus, for example, "\_traceOn" may be equivalently written "\_Traceon", "\_TRACEON", etc.

If a procedure appears in the .i.left argument of a rule; it is executed before the rule is applied. If it appears in the right argument it is executed once the rule has been applied. Trace procedures, .i.\_destru; and .i.\_stop; may appear on either side, but .i.\_goto;, .i.\_failed; and .i.\_repeat; must only appear in the .i.right argument of a rule;.

Procedures in the left argument should be placed after weights and flags, and before any variables or terminal symbols. Procedures in the right argument should be placed after variables or terminal symbols.

**8.1 Production procedures**

.i.**\_destru**; works exactly as described in §4.5, but it will only be performed if the rule in which it appears is a .i.candidate rule;.

.i.**\_goto**; is an unconditional jump to another subgrammar or another specific rule in a subgrammar. Its syntax is

\_goto(igram,irul)

where *igram* is the index of the destination grammar and *irul* the index of the destination rule. In case irul = 0, BP2 will jump to subgrammar *igram* and look for any candidate rule. If (igram,irul) does not point at a candidate rule, BP2 will either execute a \_failed conditional jump (see infra) or search a candidate rule in subgrammar *igram*. If none is found it will jump to subgrammar igram+1, etc. as usual.

When compiling the grammar BP2 will check that there is a rule indexed *irul* in subgrammar *igram*. If there is no such rule an error will be reported.

.i.**\_failed**; is a conditional jump to another subgrammar or another specific rule in a subgrammar. Its syntax is

\_failed(igram,irul)

in which *igram* and *irul* have the same meanings as in \_goto. The jump will be performed if the rule was selected by a preceding \_goto or \_failed and it was not a .i.**candidate rule**;.

.i.**\_repeat**;is a loop. Its simplest syntax is

\_repeat(n)

where *n* is an integer value indicating that the current rule should be tried *n* times. Alternate syntaxes are

\_repeat(Kx) and \_repeat(Kx = n)

where Kx is a .i.**parameter**; that may be assigned a fixed value (Kx = n) and/or controlled in real time by an external MIDI device (see QuickStart §6). Remember that fixed value assignment must be done only once in the entire grammar.

Both \_repeat and \_goto may appear in the same rule, e.g.:

X --> a b \_repeat(12) \_goto(4,8)

In this case, \_goto(4,8) will be performed once the rule has been applied 12 times or it is no longer candidate.

See example ".i.-gr.tryrepeat;" for important remarks.

**8.2 Trace procedures**

.i.**\_stop**; makes a pause. The "Resume/stop" window is displayed.

.i.**\_print**; displays the work string in the "Trace" window.

.i.**\_printOn**; and .i.**\_printOff**; set the "Display computation" mode on and off respectively. Similarly, .i.**\_traceOn**; and .i.**\_traceOff**; set the "Trace computation" mode on and off respectively, while .i.**\_stepOn**; and .i.**\_stepOff**; do the same with the "Step computation" mode.

**8.3 Renumbering and remapping conditional jumps**

If the grammar has been successfully compiled once, grammar and rule indexes have been inserted in the leftmost part of each rule. This part serves as an identification header. If you need to displace a rule, cut and paste it along with its identification header **without changing its arguments**. .i.Rule numbering;When recompiling the grammar, BP2 will renumber rules and subgrammars, and eventually modify identification headers and arguments in \_goto and \_failed procedures. Consider for example grammar ".i.-gr.tryGOTO;":

-ho.abc

-se.tryGOTO

[This grammar generates a string containing...]

[... not more than six 'c' and an equal...]

[... number of 'a' and 'b'.]

ORD

GRAM#1[1] S --> X S \_repeat(K1=12) \_goto(1,2)

GRAM#1[2] S --> lambda \_goto(2,0)

----------

RND

\_print

GRAM#2[1] <5-1> X --> C

GRAM#2[2] <1> X --> C \_goto(3,1)

----------

RND

GRAM#3[1] X --> A \_failed(4,0) \_goto(3,2)

GRAM#3[2] X --> B \_failed(3,3) \_goto(3,1)

GRAM#3[3] \_print ? --> ? B \_goto(4,0) \_print \_stop

----------

SUB

GRAM#4[1] A --> a

GRAM#4[2] B --> b

GRAM#4[3] C --> c

Suppose that rule [3] of subgrammar #3 is displaced as follows:

----------

RND

GRAM#3[3] \_print ? --> ? B \_goto(4,0) \_print \_stop

GRAM#3[1] X --> A \_failed(4,0) \_goto(3,2)

GRAM#3[2] X --> B \_failed(3,3) \_goto(3,1)

----------

After recompiling subgrammar #3 will be:

----------

RND

GRAM#3[1] \_print ? --> ? B \_goto(4,0) \_print \_stop

GRAM#3[2] X --> A \_failed(4,0) \_goto(3,3)

GRAM#3[3] X --> B \_failed(3,1) \_goto(3,2)

----------

Let us now study ".i.-gr.tryGOTO;". The first subgrammar is an "ORD" grammar therefore rules are tried in order. Rule [1] will be applied 12 times, then rule [2] will be applied so that 'S' will be erased. (Since it is controlled by K1 the number of repetitions may be later modified by a MIDI device.) The workstring will then be displayed because there is a "\_print" on top of subgrammar #2. Then rule [1] or [2] of subgrammar #2 will be applied, with a strong preference for [1] which has an initial weight <5>. Whenever rule [2] is applied BP2 will jump to rule [1] of subgrammar #3. If the rule is not candidate, the "\_failed" procedure will force it to jump to subgrammar 4. Otherwise, rules [1] and [2] will be applied alternately. If rule [2] of subgrammar #3 is no longer candidate, rule [3] will be applied once (producing an extra 'b'), then "\_goto(4,0)" will force a jump to subgrammar #4. The rest needs no explanation. It is easy to check that the numbers of 'a' and 'b' will eventually be equal.

Indeed, this is an extreme case in which there is so much procedural control that the grammar appears almost like a program (in any procedural language). Nevertheless, it demonstrates BP2's ability to combine .i.procedural knowledge;with .i.declarative knowledge;.

**9. Csound argument mapping**

When three pairs of input/output values are supplied, BP2 attempts to define a quadratic mapping from the input (x) to the output (y). Fig.20 is an example of a successful mapping.



**Fig.20 A simple quadratic mapping**

BP2 expects the mapping function to be monotonous, i.e. the three values (x1, x2, x3) supplied as the input and the ones supplied as the output (y1, y2, y3) must be strictly ordered. Even in this case a proper quadratic mapping may not be possible. See for instance Fig.21.



**Fig.21 An incorrect mapping**

Here the quadratic function is not monotonous even though the input values are ordered properly. In this case, BP2 replaces the incorrect part (the one containing an extremum) with a linear function, and changes the other part to make the curve tangent to the linear part. This yields the graph shown Fig.22.



**Fig.22 The correct mapping**

**Bibliography**

**Ames, Charles**

*Tutorial and Cookbook for COMPOSE* (revision 24 Nov. 1989)**.** Frog Peak Music, Box 9911, Oakland CA 94613, USA.

**Bel, Bernard**

*Acquisition et représentation de connaissances en musique*. PhD dissertation, Université Aix-Marseille III, 1990b. 208 pages. Microfilm available from author for US $10 (mail charges included).

*Grammaires BP pour des airs de sonneurs de cloches*. Unpublished, 1990c. 18 pages. Available from author for US $10 (mail charges included).

*Two algorithms for the instantiation of structures of musical objects*. Internal report, GRTC 458, Centre National de la Recherche Scientifique, Marseille, 1991. Available from author for US $20 (mail charges included).

Symbolic and sonological representations of sound-object structures. In *Understanding Music with AI*, M. Balaban, K. Ebcioglu & O. Laske, Eds., AAAI Press, 1992a:64-109.

Modelling improvisatory/compositional processes. *Languages of Design -- Formalisms for Word, Image and Sound* , 1, 1992b:11-26.

A flexible environment for music composition in non-European contexts. *Journées d'Informatique Musicale* (JIM 96), Caen (France), 1996a. (Available from <http://www.fortunecity.com/victorian/dada/181/jim96/jim96.htm>)

A symbolic-numeric approach to quantization in music. *3d Brazilian Symposium on Computer Music*, Recife (Brazil), 1996b.

Migrating musical concepts - an overview of the Bol Processor. *Computer Music Journal*, 22, 2, 1998:56-64.

**Bel, Bernard, & Jim Kippen**

Bol Processor grammars. In *Understanding Music with AI*, M. Balaban, K. Ebcioglu & O. Laske, Eds., AAAI Press, 1992:366-401.

**Duthen, Jacques, & Marco Stroppa**

Une représentation de structures temporelles par synchronisation de pivots. In *Le fait musical -- Sciences, Technologies, Pratiques*, eds. B. Vecchione and B. Bel. Colloque CRSM-MIM "Musique et Assistance Informatique", Marseille, October 1990.

**Kain, Richard Y.**

*Automata theory: machines and languages*. Krieger, Malabar, 1983.

**Kippen, Jim**

An ethnomusicological approach to the analysis of musical cognition. *Music Perception* 5(2), 1987:173-95.

*The Tabla of Lucknow: a Cultural Analysis of a Musical Tradition*. Cambridge University Press, Cambridge (UK), 1988.

**Kippen, Jim, & Bernard Bel**

The identification and modelling of a percussion "language", and the emergence of musical concepts in a machine-learning experimental set-up. *Computers & Humanities* 23.3, 1989a:199-214.

A pragmatic application for computers in experimental ethnomusicology. *ALLC/ICCH Conference*, Toronto (Canada), 6-10 June 1989b.

Can a computer help resolve the problem of ethnographic description? *Anthropological Quarterly*, Vol.62, N°3, 1989c:131-44.

Modelling music with grammars: formal language representation in the Bol Processor. In *Computer Representations and Models in Music*. A. Marsden and A. Pople., Eds., Academic Press, London, 1992:207-238.

Computers, Composition and the Challenge of "New Music" in Modern India. *Leonardo*, Vol.4, 1994:79-84.

**Laske, Otto**

In Search of a Generative Grammar for Music. *Perspectives of New Music*, Fall-Winter 1973, Spring-Summer 1974:351-378.

**Révész, Gyorgy**

 *Introduction to Formal Languages*. McGraw-Hill Computer Science series, New York, 1985.

**Salomaa, Arto**

*Formal Languages*. Academic Press, New York, 1973.

**Vercoe, Barry**

*Csound. A Manual for the Audio Processing System and Supporting Programs with Tutorials*. Massachusetts Institute of Technology, 1993.

Internet distribution: ftp://notam.uio.no/pub/mac/audio/

**Appendix**

**1. Solving problems**

**1.1 Using the printer serial output**

(This is only for users who did not install Opcode OMS)

The printer serial port may be used for a MIDI interface when either Appleshare is inactive or the computer is properly connected to an Ethernet or infra-red network.

The **modem** serial port is the one assumed by BP2 at startup. To set the default port to **printer**, start BP2 without loading any document and check "Printer output" it in the "Misc" menu of BP2. Then modify ".i.-se.startup;" in the "File" menu. This default settings file must be in the same folder as BP2.9.3.

BP2 does not change the output port when loading any settings file other than ".i.-se.startup;".

It is recommended to use the modem output rather than the printer output whenever possible, especially on portable machines. Opcode says that the printer output is unreliable for MIDI, it may cause the loss or incorrect timings of MIDI messages.

Some Macs have a unique serial output marked as a combined printer+modem port. BP2 sees this output as a 'modem' port.

**1.2 Inter-application communication**

In addition to exchanging .i.Apple Events;, BP2 is able to send and receive MIDI messages to/from programs running on the same computer, provided that these are OMS-compatible. Select the IAC-bus as the input or the output in the ".i.OMS input-output;" menu command.

**1.3 Memory**

BP2's memory partition is sized to 4.5 Mbytes. This determines the maximum available space for storing data generated by BP2. It requires this much to deal with large examples like ".i.-gr.ShapesInRhythm;", but it may be sized down to 1.5 Mbytes or less for small machines running small projects.

There is no limit to structures handled by BP2 (i.e. to the complexity of grammars and produced items) save for the size of its RAM partition. However, it is advisable to disable graphics when dealing with very large structures. Memory overflow on graphic procedures might still cause the program to hang, so don't forget to save data before producing sounds or graphics.

Before BP2 runs short of memory it generally sends a warning that the current task cannot be completed. It then releases some spared memory to save the current work. Should this happen, quit the application and increase

the .i.application size;: select the "BP2.9.3" icon and "Get info" in the Finder menu, then type the new value.

**2. Poem "Les Djinns"**

Tabulations have been inserted to indicate rhythm.

Murs, ville,

Et port,

A -sile

De mort,

Mer grise

Où brise

La brise,

Tout dort.

Dans la plaine

Naît un bruit.

C'est l'ha -leine

De la nuit.

El -le brame

Comme une âme

Qu'u -ne flamme

Tou -jours suit!

La voix plus haute

Sem -bl'un gre -lot.

D'un nain qui saute

C'est le ga -lop.

Il fuit, s'é -lance,

Puis en ca -dence

Sur un pied danse

Au bout d'un flot.

La ru -meur ap -proche,

L'é cho la re -dit.

C'est com -me la cloche

D'un cou -vent mau -dit;

Comme un bruit de foule

Qui tonne et qui roule,

Et tan -tôt s'é -croule.

Et tan -tôt gran -dit.

Dieu! la voix sé -pul -crale

Des Djinns! Quel bruit ils font!

Fuy -ons sous la spi -rale

De l'es -ca -lier pro -fond.

Dé -jà s'é -teint ma lampe,

Et l'om -bre de la rampe,

Qui le long du mur rampe,

Mon -te jus -qu'au pla -fond.

C'est l'es -saim des Djinns qui passe,

Et tour bil -lonne en sif -flant!

Les ifs, que leur vol fra -casse,

Cra -quent comme un pin brû -lant.

Leur trou -peau, lourd et ra -pide,

Vo -lant dans l'es -pa -ce vide,

Sem ble un nu -a -ge li -vide

Qui port' un é -clair au flanc.

Ils sont tout près! Te -nons fer -mée

Cet -te salle, où nous les nar -gons.

Quel bruit de -hors! Hi -deuse ar -mée

De vam -pi -res et de dra -gons!

La pou -tre du toit des -cel -lée

Ploie ain -si qu'une her -be mouil -lée,

Et la vieil -le por -te rouil -lée,

Trem -ble à dé -ra -ci -ner ses gonds.

Cris de l'en -fer! voix qui hurle et qui pleure!

L'hor -rible es -saim, pous -sé par l'a -qui -lon,

Sans doute, ô ciel! s'a -bat sur ma de -meure.

Le mur flé -chit sous le noir ba -tail -lon..

La mai -son crie et chan -cel le pen -chée,

Et l'on di -rait que, du sol ar -ra -chée,

Ain -si qu'il chasse u -ne feuil -le sé -chée,

Le vent la roule a -vec leur tour -bil -lon!

Pro -phè -te! si ta main me sauve

De ses im -purs dé -mons des soirs,

J'i -rai pros -ter -ner mon front chauve

De -vant tes sa -crés en -cen -soirs!

Fait que sur ces por -tes fi -dèles

meu -re leur souf -fle d'é -tin -celles,

Et qu'en vain l'on -gle de leurs ailes

Grince et crie à ces vi -traux noirs!

Ils sont pas -sés. Leur co -horte

S'en -vole et fuit, et leurs pieds

Ces -sent de bat -tre ma porte

De leurs coups mul -ti -pli -és.

L'air est plein d'un bruit de chaînes

Et dans les fo -rêts pro -chaines

Fris -son -nent tous les grands chênes

Sous leur vol de feu pli -és!

De leurs ai -les loin -taines

Le bat -te -ment dé -croît,

Si con -fus dans les plaines,

Si fai -ble, que l'on croit

Ou -ïr la sau -te -relle

Cri -er d'u -ne voix grêle,

Ou pé -til -ler la grêle

Sur le plomb d'un vieux toit.

D'é -tran -ges syl -labes

Nous vien -nent en -cor;

Ain -si, des a -rabes

Quand son -ne le cor,

Un chant sur la grève

Par ins -tants s'é -lève,

Et l'en -fant qui rêve

Fait des rê -ves d'or.

Les Djinns fu -nèbres,

Fils du tré -pas,

Dans les té -nèbres

Pres -sent leurs pas;

Leur es -saim gronde:

Ain -si, pro -fonde,

Mur -mure une onde

Qu'on ne voit pas.

Ce bruit vague

Qui s'en -dort,

C'est la vague

Sur le bord;

C'est la plainte,

Pres -qu'é -teinte,

D'u -ne sainte

Pour un mort.

On doute

La nuit...

J'é -coute:

Tout fuit,

Tout passe;

L'es -pace

Ef -face

Le bruit.

**3. Disclaimer**

BP2 is a program producing music under MacOS. The program is provided "as is", and the author makes no warranty, either express or implied, with respect to this software, its performance or suitability for any particular purpose. If you use the BP2 program, you do so at your own risk. The author disclaims all liability for loss of data, mechanical damage, or other losses suffered while using the BP2 program. By using the software you automatically agree to these terms.

If you don't agree with this policy, or laws in your country do not permit such a disclaimer, permission to use the program is NOT granted.

**4. Acknowledgements**

I feel indebted to Prof. Jim Kippen (University of Toronto) for initiating a fascinating musical and scientific venture when he prompted me to design a "word-processor" for rhythmic material in 1981. The years during which we developed BP1 and related tools have undoubtedly determined our present careers.

My wife Andréine, a dancer and choreographer who studied under Kathak maestro Pt. Birju Maharaj for about 18 years, deserves special thanks for her patience in seeing me spend days and nights on the computer.

Our teen-ager son Tansen, a wizzard in 3D animation, keeps on looking down at BP2's interface and questions its (my) ability to deal with interactive graphics, QuickTime objects, virtual reality and the like. He was nevertheless impressed by scripts, new debugging techniques, and he even started liking some of the sound examples! He made no comment when he saw QuickTime Music running under OMS.

**Index**

+ sc.<name> 32

+sc.checkAll 67

+sc.TryMe 11; 13

--ho.abc1 33

-cs.<name> 32; 87; 92

-cs.tryCsound 88

-cs.tryCsoundObjects 94; 95; 97

-da.<name> 32

-da.checkArticulation 17

-da.checkCapture 16

-da.checkControls 14; 15

-da.checkPoly 22

-da.checkTimeSet 27; 57; 82

-da.Csoundtest 93

-da.dhin-- 75

-da.kathak tihais 76

-da.lahras 77

-da.makePrototypes 27

-da.PeriodNotation 19

-da.preroll 29

-da.ShowPeriods 21; 74

-da.SomeNotes 14

-da.tryCsound 91; 98

-da.tryCsoundInstruments 99

-da.tryCsoundObjects 94; 97

-da.tryQuantize 127

-da.trytags 64

-da.tryTicks 48

-da.tryTimePatterns 70

-gl.<name> 32

-gl.D50 66

-gl.GeneralMIDI 14; 66

-gr.12345678 77

-gr.<name> 32

-gr.acceleration 120

-gr.bells 53; 121

-gr.cloches1 29; 53; 107

-gr.dhadhatite 76

-gr.dhati 76

-gr.dhin-- 29; 73; 74; 75

-gr.Djinns 49

-gr.doeslittle 36

-gr.gramgene1 122

-gr.gramgene2 122

-gr.koto3 33; 37; 42; 55; 57; 58; 61; 62; 84; 123

-gr.kss2 77

-gr.Mozart 53; 58; 84; 109

-gr.MozartExpression 71

-gr.polyphony1 46

-gr.Ruwet 29; 107

-gr.ShapesInRhythm 77; 137

-gr.trial.mohanam 77

-gr.tryDESTRU 111

-gr.tryflags 77

-gr.tryflags2 78

-gr.tryflags3 79

-gr.tryGOTO 132; 133

-gr.tryhomomorphism 28

-gr.tryLIN 109; 111

-gr.tryrepeat 132

-gr.tryTimePatterns 48; 70

-ho.<name> 32; 107

-ho.abc1 33

-ho.tryhomomorphism 28

-in.<name> 32; 59

-kb.<name> 32

-mi.<name> 32; 88

-mi.abc1 33

-mi.tryCsoundObjects 94

-se.<filename> 59

-se.<name> 32

-se.startup 9; 10; 18; 85; 137

-tb.<name> 32; 48

-tb.slowshift 47

-wg.<name> 32

1.12.2 Polymetric expression 21

1.7 Entering the arguments of performance controls from MIDI 16

17.6 Csound function tables describing waveforms 93

2.3.2 Capturing from text 27

4.10 Polymetric expression 115

4.11 Period notation 120

4.5 Destroying structure 111

accurate transcription of durations 48

additional Csound instrument parameters 91

ADDval 91

"Allow randomize" button 37

alphabet file 37; 106

"Alphabet" window 28; 107

Apple Event (receiving) 65

Apple Event (sending) 66

Apple Events 65; 66; 137

application size 138

"Autorandomize" button 54

Bol Processor BP1 130

BP2 help 9

BP2 score 27; 87

BP2.9.3 (68k) 9

BP2.9.3 (ppc) 9

BP2test.orc 92; 93

BP2test.sco 93

bracketing 71; 109

Break tempo 125

BrkTempo 105

Buffer size 53

candidate rule 81; 131; 132

canonic solution 126

Capture selection as... 27

Capture settings 48

cellular automata 123

Change instrument file 94

chegun 73

Choose candidate rule 79

chunk rule 130

client-server technology 65

Compilation 38

compiling a glossary 40

complete polymetric expression 46; 102

computation time 63

Compute while playing 56

"Compute while playing" button 36

concatenation symbol 114

condition flag 77

conditional jumps 39

constraint propagation 125

ContBeg 105; 126

ContEnd 105; 126

"Continue" MIDI message 56

continuity constraint 126

Control Panel 37; 41

control parameter 17; 63; 91; 93

Convert to Csound 27; 95

cps (Hz) 53

Cscore 86

Csound 87

"Csound instruments" dialog 88

Csound output 92

Csound score 56

Csound scores 8

Csound scores in sound-object prototypes 94

Csound waveform description 93

Cyclic play 57

"Data" window 41

decision file 81

declarative knowledge 131; 133

default orchestra file 92

default positioning of objects 125

density of sound-objects 20

derivation mode (grammar) 109

derivation mode (rule) 111

Destroying structure 130; 131

dhamar 74

diapason frequency 53; 90

dilation ratio 25; 30; 104

Discard NoteOff's except in last period 30

Display time setting 82; 126

Don't stop 53

Don't strike again NoteOn's 30

DOS or Windows editors 36

dynamic weight 49; 56; 112

Dynamic weight assignment 130

Edit/record MIDI stream 95

empty string 108

Erasing rule 130

event table 117; 118

Expand display 45

explicit tempo marker 20; 73; 113

export music 8

file saving options 82

file types 32

FixScale 104

flag conditions 78

flags 77

Force integer number of periods 30

fractional gap 114

GEN07 91

GEN08 91

General MIDI 15; 66

global drift 125

glossary 14; 66

glossary as a script 67

grammar consistency 40

grammar procedure 39

Graphic scale 44

hierarchy of time-span intervals 70

homomorphic transformation 109

homomorphism 28; 29; 32; 38; 53; 107

Ignore constraints 56

Import MIDI file 27

inactive rule 112

inference engine 102

infinite weight 113

Initialisation 57

input sound-object 64

Insert silence 25

insertion point 24; 25

instanciating a sound-object 123

Inter-Application Communication (IAC) 7

"Interaction" window 59

Interactive 56

Interactive (use input) 59

"Interactive (use input)" button 59

interactive file 37

interactive mode 56

interpretation of .i.polymetric expression 102

irregular beats 69

join-semilattice 70

Kathak dance 76

keyboard encoding 75

Learn weights 75

LEFT 111

left argument of a rule 40; 66; 108; 121; 131

legato 17

LIN 109; 111

Load decisions 81

Load time-pattern 72

loading any file 37

local drift 125

Lucknow style of tabla 77

measured smooth time 69

membership test 74; 75; 130

metagrammar 122; 130

metavariable 113

metrical properties of sound-object 126

"Metronom" dialog 47; 73

metronome value 46

MIDI file 48; 56; 82

MIDI file formats 82

MIDI files 8

MIDI orchestra 14

MIDI orchestra file 37

"MIDI PANIC" button 37

MIDI properties 106

MIDI sampler 48

MIDI set-up time 52

MIDI torture 14

Modify '-se.startup' 11; 85

MULTval 91

musical dice game 58

musical gesture 103

New seed 54

non-measured smooth time 73

"Non-stop improvize" button 36

note convention 18

note prolongation 13

number tihai 77

Object concatenation 115

OkCompress 104

OkExpand 104

OkRescale 104

OMNI OFF 63

OMNI ON 11

OMNI ON/OFF 15

OMS input-output 10; 11; 137

OMS MIDI setup 10; 11

OMS Preferred Device 10

OMS Setup 9

OMS Studio setup 10

on-setting time 31

Opcode OMS 7

Open Music System 7

optimisation of computational space 129

or.<name> 32

ORD 109; 111

organum 125

out-time sound-object 38; 46; 65

"Output" menu 88

OverBeg 105; 126

OverEnd 105; 126

overlapping constraint 126

parallel derivation 123

parameter 132

parsing 130

pattern 108

pattern grammar 108; 111

pattern language 108

PATTERNS 70

performance control 13; 14; 16; 17; 46; 57; 58; 64; 77

period notation 19; 22; 48

Periodical sound-object 28; 104

periods 73; 74

phase diagram 22; 46; 51; 115; 116; 124; 127; 129; 130

physical duration of sound object 69; 125

Pick performance control 14; 16

PivBeg 103

PivBegOn 103

PivCent 103

PivCentOnOff 103

PivEnd 103

PivEndOff 103

pivot 25; 44

pivot location 25

PivSpec 103

play all substitutions 123

Play selection 88

Play ticks 47; 48

POLY/MONO 15

polymetric expansion algorithm 22

polymetric expression 13; 15; 21; 22; 45; 46; 58; 59; 64; 67; 70; 74; 84; 102; 115; 116; 117; 127; 130

polymetric representation 120

polyphonic music 115

polyrhythmic pattern 49

post-roll 30; 31

pre-roll 30

procedural knowledge 131; 133

procedure 131

Produce items... 42

production procedures 77; 78; 130

production rule 108

programmed grammar 56; 77; 130; 131

prolongation 38

qa'ida 73; 77

quantization 51; 52; 128; 129

QuickTime Music 8; 9; 10; 14

QuickTime® Musical Instruments 8

random decisions 54

random numbers 54

random seed number 54

random-sequence generator 54

Randomize 54

REC 12

Receive MIDI data to file 83

reference expression 109

reference tempo 25

Reloc 105; 126

relocatable object 44

remote context 121

Repeat computation 81

Reset flags 56

Reset rule weights 56

Resume 42

RIGHT 111

right argument of a glossary rule 67

right argument of a rule 40; 108; 130; 131

RND 109; 111

Rule numbering 39; 132

Run MIDI in background 11

Save decisions 81

Save settings 85

"Scrap" window 83

"Script" dialog 12

"Script" menu 12; 61

"Script" window 12

self-imbedding grammar 79

self-similar fractal 109

Send dilation ratio to controller 105

Send MIDI data from file 83

sequentiality of sound-objects 129

Set tempo 63

"Settings" dialog 26; 36; 54; 55; 59

settings file 37; 85

Show graphics 126

Show periods 74

silence 13; 38; 113

Silences 21

simple note 13; 22; 106

Simple notes 28

slave expressions 109

smooth time 46; 48; 69

sound-object 13; 22; 69; 106

Sound-object duration 31

sound-object file 43

sound-object properties 25

sound-object prototype 24; 25; 102; 103

sound-object prototype editor 22; 23; 25

sound-object structure 43; 44; 46

special line break 84

staccato 17

"Start" MIDI message 56

Start string 40; 41; 108

"Start string" window 40

starting symbol 36; 108

Step time setting 126

Step-by-step compute 81

Stop 42

streak 44

striated time 68

structural marker 74

structure of time 124

SUB 58; 109; 123

SUB1 58; 109

subgrammar 109

Substitution 123; 130

symbolic duration 25; 46; 71; 114

symbolic durations 19

symbolic tempo 113

Symbolic time 102

synchronisation tag 64

Synchronize start 56

synthesis of tabla 77

template 74; 75

tempo 20

terminal symbol 28; 106; 107

"Tick settings" dialog 48

ticks 47

tigun 74

tihai 77

Time accuracy 52

"Time base" dialog 28; 46; 47; 73

time base file 37

time pivot 123; 124

time quantization 127

Time resolution 52

time setting 102

time setting algorithm 26; 56; 125; 126

Time setting display 57

Time setting step 57

Time setting trace 57

time streak 25; 26; 69; 71; 103; 124; 125; 127; 129

time structure 44

time-object 69

time-pattern 70; 71; 72

time-patterns in polymetric expression 70

time-span interval 30; 31; 124

tintal 73

Tokenised alphabet 38

tokenised alphabet/grammar/glossary 38

Tokenised grammar 39

tonal transformations 107

topological properties of sound-object 126

Trace Csound 57; 87; 94

"Trace" window 38; 39; 83

transformational grammar 109

Transpose input notes 13

Transposition 15; 53

tree structure 70

true BP grammar 74; 130

TruncBeg 105

TruncEnd 105

TryAppleEvents.fm 11; 65

"Tuning" dialog 53

Type from MIDI 13; 19; 59; 62

Types-creators 32

undefined variable 40

undetermined rest 21; 117

Undo 81

unreachable variable 40

Use each substitution 55; 57

Use MIDI in/out 56; 59

using any expression as a start string 41

variable 57; 107

Velocity 15

weight inference 75

weights in grammar 17; 39; 58; 64; 75; 102; 112

wild card 113

window size 83

work string 36; 53

Write Csound scores 56; 93

Write MIDI files 56

\_chan 16

\_cont 15; 91

\_destru 112; 131

\_failed 131; 132

\_fixed 15; 91

\_goto 131

\_legato 17

\_mod 16

\_pancontrol 16

\_pitchbend 16

\_pitchrange 16

\_print 131; 132

\_printOff 131; 132

\_printOn 131; 132

\_repeat 131; 132

\_rest 21

\_script(MIDI program x) 14

\_staccato 17

\_step 15; 91

\_stepOff 131; 132

\_stepOn 131; 132

\_stop 131; 132

\_traceOff 131; 132

\_traceOn 131; 132

\_transpose 15; 53

\_value 15; 91

\_vel 16; 17

\_velcont 15

\_velfixed 15

\_velstep 15

\_volumecontrol 16