Journal of Magnetic Resonance 204 (2010) 189–195

Contents lists available at ScienceDirect

Journal of Magnetic Resonance

journal homepage: www.elsevier.com/locate/jmr

Pulse sequence editing by symbolic calculation

Michael P. Barnett [a,*], Istvàn Pelczer [b]

[a] Meadow Lakes, Hightstown, NJ 08520, USA
[b] Department of Chemistry, Princeton University, Princeton, NJ 08544, USA

ARTICLE INFO

Article history:
Received 8 July 2009
Revised 10 January 2010
Available online 22 January 2010

Keywords:
Pulse sequence diagrams
Symbolic calculation
Computer algebra
CA
Mathematica
Modeling

ABSTRACT

Our APSEQ software package interprets commands to alter, combine and draw NMR pulse sequences that are expressed in a simple mnemonic style. The package is coded in MATHEMATICA.[†] It can be extended to meet specialized needs.

1. Introduction

Hundreds of pulse sequences have been proposed, analyzed and used in the many kinds of NMR experiments developed over the years. The numbers continue to increase. The modularity of pulse sequences invites the application of symbolic calculation to their construction. MATHEMATICA and MAPLE have been used in product operator analysis [1, 2, 3], and in other NMR work listed in [4].

Here, we report our APSEQ package that focuses on the representation, modification and combination of pulse sequences, and the construction of pulse sequence diagrams. We are exploring interfaces with packages for product operator analysis. APSEQ is built on the principles of the MATHSCAPE package that has been used primarily to construct mathematical formulas for chemical computations, and in studies of mathematical reasoning [5, 6]. APSEQ abbreviates "analogy in pulse sequence (construction)".

§2 and §3 list the APSEQ conventions systematically, by reference to examples of their usage. §4 draws attention to the principal MATHEMATICA and MATHSCAPE commands that facilitate the construction of large sets of inter-related pulse sequences from concise input. §5 discusses possible extensions of this work. Supplementary material explains further examples [7], altering details of the diagrams [8], and the implementation [9]. The distribution file contains the APSEQ software and control files to check its operation. [10].

*Corresponding author
Email addresses: michaelb@princeton.edu (Michael P. Barnett), ipelczer@princeton.edu (Istvàn Pelczer)

[†] MATHEMATICA is a registered trademark of Wolfram Research Inc.

0042-6989/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jmr.2010.01.009

2. The basic APSEQ conventions

APSEQ notation represents, overall, a sequence of events that may consist, individually, of concurrent subsidiary events. The refocused HETCOR sequence (Fig. 2.1) provides a simple illustration. To experiment with the software in a UNIX environment, type `math` to start a MATHEMATICA session. Then type `<<apseq.m` to load the software. The next statement produces Fig. 2.1.

```
draw[pulseSeq[HETCOR, refocused] =
 {channels[H1, C13], {pulse[90], }, delay[t1/2],
  {, pulse[180]}, delay[t1/2], delay[1/(2J)],
  pulse[90], delay[1/(3J)],
  {decouple, acquire}[taq]}]
```

In MATHEMATICA, square brackets `[ ]` contain arguments of a function or qualifications of a name, and curly brackets `{}` contain the elements of a list. APSEQ converts the `draw` statement into a pdf file that is brought into a LaTeX document, such as the present paper, by an `\includegraphics` command. Hence Fig. 2.1.

This example illustrates the following conventions.

C1: A pulse sequence is represented by a list of items that specify pulses, delays and other components.

C2: A `channels[...]` item usually begins the list. This provides names for the channels and, implicitly, how many there are. APSEQ recognizes mnemonics for isotopes and other chemical entities *e.g.* `H1`, `C13` and `Calpha13` for ^{1}H, ^{13}C and $^{13}C^{\alpha}$, and `Gz` for the gradient channel.

C3: `delay[`t`]` specifies a delay t. The symbols `t1`, `t2`, ..., are displayed as t_1, t_2, ... Greek letters are specified by name, *e.g.* `pi`, `phi` and `Delta`. Subscripts are typed immediately after these, *e.g.* `taum` for τ_m.

C4: The pulse β delivered concurrently on all channels with phase ϕ is denoted by the expression `pulse[`β,ϕ`]` at the

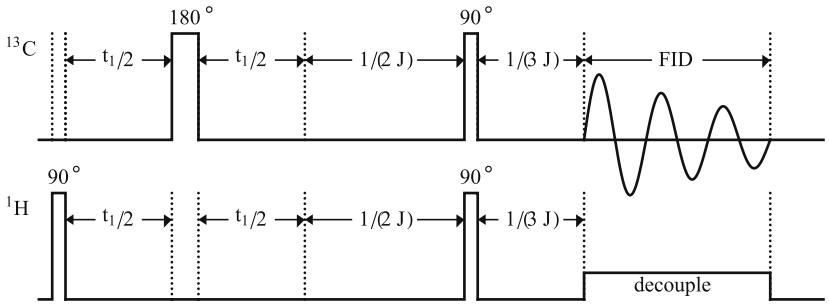

Fig. 2.1. Refocused HETCOR.

outermost level of the overall list. The 7^{th} item in the pulse sequence under current consideration is an example. The phase can be explicit $\pm x$, $\pm y$ or, in a phase cycled sequence, symbolic, *e.g.* ψ.

C5: Different concurrent events on different channels are listed between curly brackets *e.g.* `{pulse[90], }` and `{, pulse[180]}` represent a 90° pulse on channel 1 and a 180° on channel 2, respectively, with null action in the channel that is not receiving a pulse. Correspondingly, concurrent 90°_y and 90°_x pulses on the I and S channels are denoted by `{pulse[90, y], pulse[90, x]}`. In general, different concurrent events for an *n*-channel experiment are depicted by a list of *n* items, some of which may be null, between curly brackets.

C6: Further common elements of pulse diagrams are denoted by `acquire`, `decouple`, `spinLock` and `gradientPulse`. The repertoire is being extended.

C7: Square brackets, following a list of concurrent events, enclose their common duration, as in `{decouple, acquire}[taq]`.

C8: APSEQ provides defaults for the relative placement of the channels, and the lengths of the lines that depict pulses, delays and other events. These can be altered by assigning new values [8].

C9: The statement `draw[pulseSeq[`*name*`] = {...}]` assigns *name* to the list that represents the pulse sequence, as well as drawing it.

C10: This single statement can be split into the two statements `pulseSeq[`*name*`] = {...}; drawPulseSequence[`*name*`]`

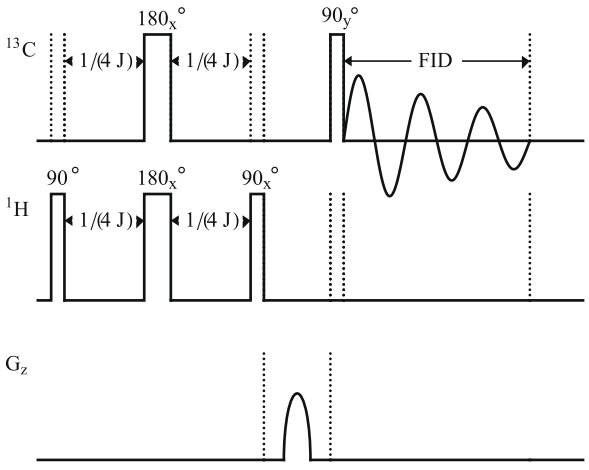

Fig. 2.2. Gradient assisted INEPT.

C11: The output is a POSTSCRIPT file when MATHEMATICA release 5.2 or earlier is used. It is a PDF file when release 6 or 7 is used. The file is called *name′*`.ps` or *name′*`.pdf` in these two cases, where *name′* is formed from *name* by eliding spaces and substituting `_` for each comma.

Fig. 2.2 is produced by the next statement. It illustrates convention **C12**.

```
draw[pulseSeq[INEPT, gradientAssisted] =
  {channels[H1, C13, Gz], {pulse[90], },
   delay[1/(4J)], pulse[180, x], delay[1/(4J)],
   {pulse[90, x],}, gradientPulse[5 s],
   {, pulse[90, y]},{, acquire}[taq]}]
```

C12: `gradientPulse[`*g*`]` specifies a gradient pulse with strength *g*. Putting `s` after the strength when the precise value does not matter suppresses the label.

Fig. 2.3 is produced by the next statement, covered by **C1**–**C4** and **C6**. It is the basis of several further examples in §4.

```
draw[pulseSeq[2D, TOCSY] =
  {channels[H1], pulse[90], delay[t1],
   spinLock[taum], acquire[t2]}]
```

C13: Including the argument `notes[`$s_1, s_2, \ldots$`]` at the end of a `drawPulseSequence` expression displays the character strings s_i on successive lines at the foot of the diagram. Thus, Fig. 2.4 is produced by

```
pulseSeq[2D, HSQC, basic] =
  {channels[H1, N15], {pulse[90], }, delay[Delta],
   pulse[180], delay[Delta], pulse[90], delay[t1/2],
   {pulse[180], }, delay[t1/2], pulse[90],
   delay[Delta], pulse[180], delay[Delta],
   {acquire, decouple}[t2]}

drawPulseSequence[2D, HSQC, basic,
   notes["Delta = 1/(4J), J = J[1, NH]"]]
```

C14 (phase cycling): To explain the APSEQ commands for phase cycling, we consider the pulse sequence

```
pulseSeq[short] =
 {channels[H1], pulse[90,phi], delay[t], acquire[taq]}
```

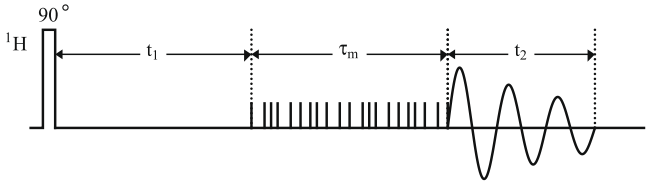

Fig. 2.3. Basic 2D TOCSY.

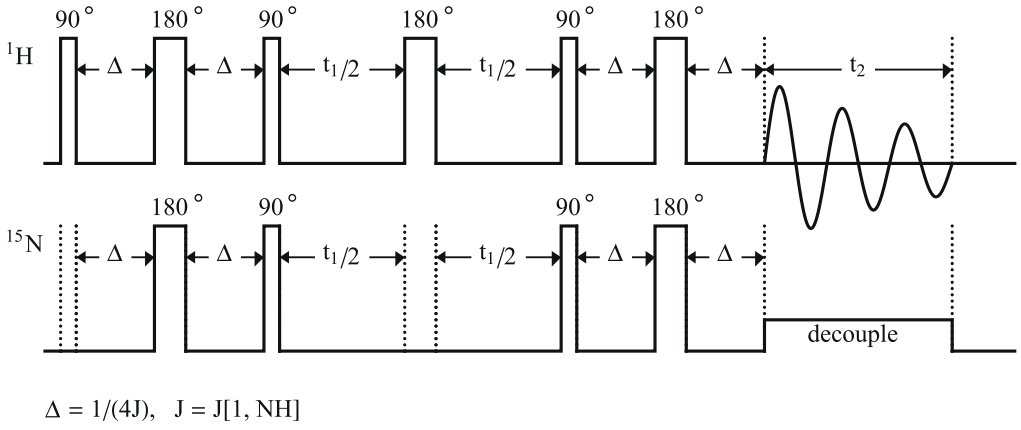

Fig. 2.4. Basic 2D HSQC.

The MATHEMATICA language supports expressions such as `f[x1, x2][y1, y2, y3]`, often called "Curry notation". We represent a phase cycled experiment, in this notation, by `phaseCycle["..."][...]`, where the first ... stands for the phases in successive cycles, and the second ... stands for the pulse sequence. We use a name of the form `phaseCycled[...]` for the complete expression, as in the second statement that follows

```
phaseList =
  "phi = x, y, -x, -y; receiver = -y, x, y, -x"

phaseCycled[explanatory] =
    phaseCycle[phaseList][pulseSeq[short]]
```

The names in square brackets, in this case `explanatory`, `phaseList` and `short`, are the user's choice. The statement `drawPhaseCycled[explanatory]` draws the pulse sequence and displays the cycling information at the foot of the diagram. The `expand[]` function makes the multi-step sequence explicit. Thus

```
phaseCycled[explanatory] // expand[]    =>
{channels[H1], pulse[90, " x"], delay[t],
 acquire[taq], pulse[90, " y"], delay[t],
 acquire[taq], pulse[90, " -x"], delay[t],
 acquire[taq], pulse[90, " -y"], delay[t],
 acquire[taq]}
```

In MATHEMATICA, *x*//*f* is an alternative way of writing *f*[*x*]. Repetition factors and implied repetition can be used in the cycling specification, as in the 32-step phase cycle for the ^{1}H–^{13}C CT–HSQC experiment.

```
forCTHSQC =
  "phi1 = x, -x; phi2 = 8(x), 8(-x);
   phi3 = 2(x), 2(y), 2(-x), 2(-y);
   phi4 = 16(y), 16(-y);
   receiver = 2(x, -x, -x, x), 2(-x, x, x, -x)"
```

The expression `phaseCycled[...]//expand[n]` truncates the experiment to *n* cycles. Omitting *n* gives the complete expansion, as above. *n* should be a multiple of 4. Explicit phase cycle specifications can be altered by the simple list editing commands (see **C20**).

3. The editing commands

C15 (Simple replacement): In MATHEMATICA, *u* `/.` *a* `->` *b* substitutes *b* in place of every occurrence of *a* in *u*. Consider the COSY-β module of a phase cycling sequence

```
pulseSeq[COSY, beta] =
   {pulse[pi/2, phi1], delay[t1], pulse[beta, phi2]}
```

This is made specific to the COSY–35 sequence by

```
pulseSeq[COSY, 35] =
 pulseSeq[COSY, beta] /. beta -> 35
```

Correspondingly, `decouple` is made more specific by `/. decouple -> GARP` and `/. decouple -> WALTZ16`.

C16 (Multiple replacement): In MATHEMATICA, *u* `/. {`a_1 `->` b_1, a_2 `->` $b_2, \ldots$ `}` substitutes b_1 in place of every occurrence of a_1, b_2 in place of every remaining a_2 and so on. For example, in the implementation of the phase cycling notation, *unitOfRepeat* `/. {phi1->x, phi2->-x, receiver->x}` sets the specified phases to $x, -x, x$,

C17 (Sequence replacement): In the MATHSCAPE conventions, *u* `/.` *a* `->` `sequence[`*p, q,*...`]` replaces each occurrence of *a* by the specified sequence, with the punctuation shown by the prototypes:

```
{a,b,c} /. b -> sequence[p,q]  => {a,p,q,c}
{a,b,c} /. a -> sequence[p,q]  => {p,q,b,c}
{a,b,c} /. c -> sequence[p,q]  => {a,b,p,q}
```

This feature is used to put a sandwich pulse in place of the inversion pulse in a DEPT experiment by

```
pulseSeq[DEPT] /. pulse[180,x] ->
   sequence[pulse[90,x], pulse[180,y], pulse[90,x]]
```

C18 (Insertion): The basic 2D HSQC pulse sequence, shown in Fig. 2.4, is converted to the corresponding TOCSY sequence, shown in Fig. 3.1, by inserting a TOCSY spin lock on the proton channel before acquisition. The next two statements construct the TOCSY sequence and draw it, respectively.

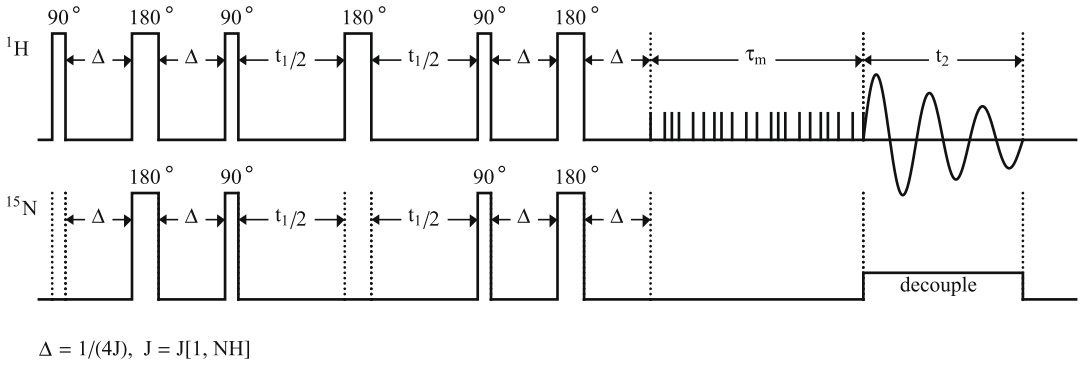

Fig. 3.1. Basic 2D HSQC TOCSY.

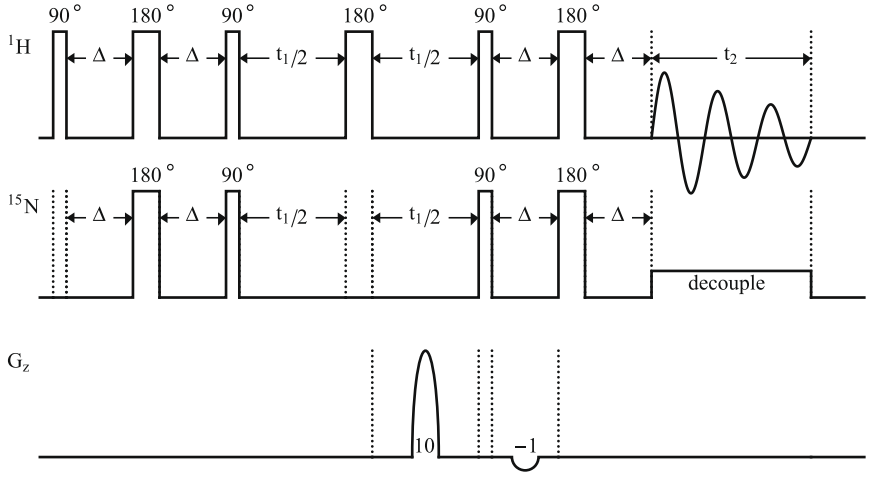

Fig. 3.2. 2D gradient selected HSQC.

```
pulseSeq[2D, HSQC, basic, TOCSY] =
  pulseSeq[2D, HSQC, basic] //
   insert[spinLock[taum], on[H1], before[acquire]]

drawPulseSequence[2D, HSQC, basic, TOCSY,
  notes["Delta = 1/(4J),  J = J[1, NH]"]]
```

APSEQ supports several `insert` commands that are covered by the syntax

`insert[`$u_1, u_2, \ldots$`,`
`on[`*channelName*`(`s`)], {before|in|after}[`*key*`,` *count*`]][`t`]`

Each u_i is either an elementary pulse sequence item (*e.g.* `pulse[...]`), or `spinEcho` (the repertoire of such higher level elements is being extended), or a MATHEMATICA replacement *a*`->`*b*. The `count` can be omitted when it is 1. Also, the channel specification G_z can be omitted when inserting a gradient pulse.

C19 (Composition): The command `pipe[`$u_1, u_2, \ldots$`]` carries out the actions $u_1, u_2, \ldots$, in succession. Consequently, Fig. 3.2 is produced by:

```
pulseSeq[2D, HSQC, gradientSelected] =
 pulseSeq[2D, HSQC, basic] //
  pipe[
   insert[gradientPulse[10], in[delay[t1/2], 2]],
   insert[gradientPulse[-1], in[delay[Delta], 3]]]

drawPulseSequence[2D, HSQC, gradientSelected,
   notes["t1/2 > gradient pulse"]]
```

C20 (Simple list editing): When m is an integer, the MATHSCAPE command `insert[`u, m`][`ℓ`]` inserts u at position m in ℓ, and `delete[`m`][`ℓ`]` deletes the m^{th} element of ℓ. Several more list editing commands are described in [5].

C21 (Pulse trains): The representations of composite pulses are generated by elementary MATHEMATICA commands that operate on lists, wrapped in simple APSEQ functions. `attachCopy` converts $\{\ell_1, \ldots, \ell_n\}$ to $\{\ell_1, \ldots, \ell_n, \ell_1, \ldots, \ell_n, \}$. `attachInverse` converts $\{\ell_1, \ldots, \ell_n\}$ to $\{\ell_1, \ldots, \ell_n, -\ell_1, \ldots, -\ell_n, \}$. `coalesce` converts consecutive $\pm1, \pm2$ to ±3 and $\pm3, \pm1$ to ±4. WALTZ-4, 8 and 16 are produced, using these, by

```
RRRbRb = {R} // attachCopy // attachInverse

waltz4 = RRRbRb /. R -> {1, -2, 3} //Flatten

Rp = waltz4 // RotateLeft // coalesce

waltz8 = Rp // attachInverse

waltz16 =
 waltz8 // RotateRight // attachInverse // coalesce
```

Actions that join (i) a list and (ii) a result of modifying its elements by symmetry operations, are of key importance in Levitt's construction of multiple pulses by analogy to frieze patterns [11].

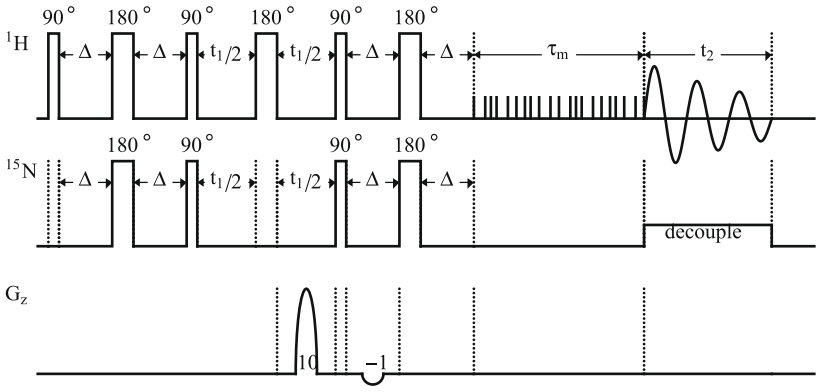

Fig. 4.1. 2D gradient selected HSQC TOCSY.

4. Constructing a batch of pulse sequences

C22 (Analogy): There is an extensive and growing literature on the recognition and use of analogy in mathematics, science and the humanities. The particular aspect that we use in pulse sequence construction is illustrated by the next statement. The infrastructure is discussed in [5, 6].

```
adaptAssignmentOfPulse[2D, HSQC, basic, TOCSY,
 by[basic -> gradientSelected]]
```

This produces the following intermediate result, and the statement after it produces Fig. 4.1.

```
{channels[H1, N15, Gz], {pulse[90], Null},
 delay[Delta], pulse[180], delay[Delta], pulse[90],
 delay[t1/2], {pulse[180], Null},
 {delay, delay, gradientPulse[10]}[t1/2], pulse[90],
 {delay, delay, gradientPulse[-1]}[Delta],
 pulse[180], delay[Delta],
 {spinLock, Null, Null}[taum],
 {acquire, decouple}[t2]}

drawPulseSequence[2D, HSQC, gradientSelected, TOCSY]
```

C23 (Mapping): The command that produced Fig. 4.1 is extended to the production of the variations of 2D HSQC TOCSY that correspond to three variations of HSQC, by two further features of MATHEMATICA called "pure functions" and "mapping" [12]. These are combined in the prototype statement

```
f⌈#⌉& /@ {a. b. c} => {f⌈a⌉. f⌈b⌉. f⌈c⌉}
```

The `f[#]&` acts as a template for the successive items of a list. These are constructed by substituting the successive items in the list `{a, b, c}` in place of the `#` and dropping the `&`. A MATHEMATICA expression that contains `#` and has `&` at the end is called a "pure function". It corresponds to a λ-expression in mathematical logic, that is often called an "anonymous function". The operation performed by the `/@` symbol is called "mapping". The pulse sequence statement that is described next is also a "compound statement", *i.e.* a succession of statements, that can stand alone, contained in parentheses. An elementary example of mapping is

```
(f[#] = #+2; Print[f[#]])& /@ {x, y, z}
```

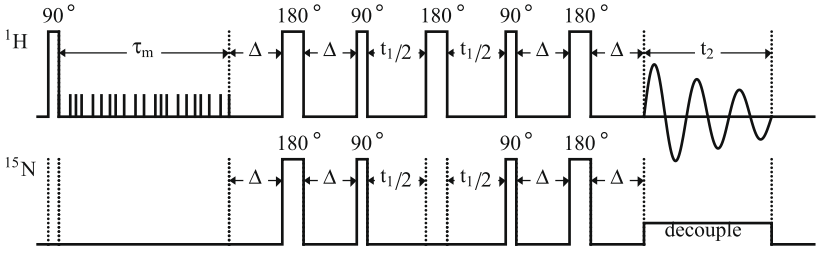

Fig. 4.2. Basic 2D TOCSY HSQC.

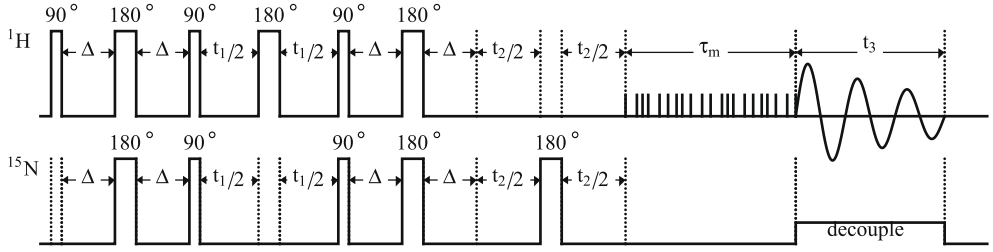

Fig. 4.3. Basic 3D HSQC TOCSY.

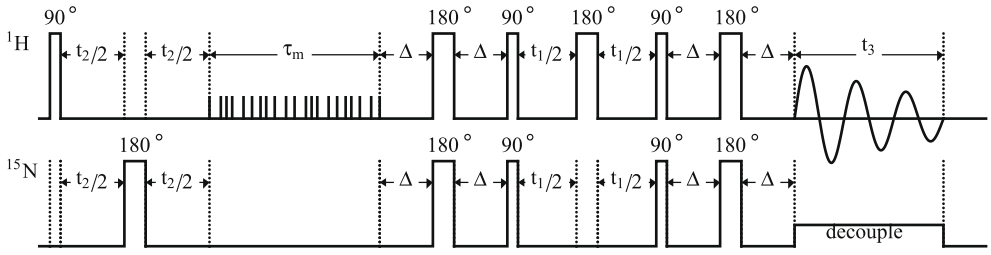

Fig. 4.4. Basic 3D TOCSY HSQC.

This prints x+2, y+2, and z+2 on three successive lines. In a similar fashion, the representations and the POSTSCRIPT files for three variations on the basic 2D HSQC TOCSY are produced by

```
(adaptAssignmentOfPulse[2D, HSQC, basic, TOCSY,
  by[basic -> #]];
 drawPulseSequence[2D, HSQC, #, TOCSY])& /@
 {gradientSelected, editedGradientSelected,
  gradientEnhanced}
```

This reproduces the gradient selected pulse sequence shown in Fig. 4.1 and the APSEQ expressions and the diagrams for the edited gradient selected and the gradient enhanced pulse sequences shown in [7], which gives all the relevant details.

The combination of the pure function, map and compound statement features of MATHEMATICA open up further ways to systematize the construction of pulse sequence diagrams. Thus, the next two statements construct the 2D TOCSY HSQC pulse sequences that correspond to the HSQC TOCSY variations that are considered above.

```
hsqcList =
 {basic, gradientSelected, editedGradientSelected,
  gradientEnhanced}

(pulseSeq[2D, TOCSY, HSQC, #] =
  pulseSeq[2D, HSQC, #] //
   insert[spinLock[taum], on[H1],
    before[delay[Delta], 1]];
 drawPulseSequence[2D, TOCSY, HSQC, #])& /@ hsqcList
```

Hence Fig. 4.2 for the basic 2D TOCSY HSQC pulse sequence and three variants shown in [7]. The prescriptions for the 2D HSQC TOCSY sequences are converted to 3D by changing t_2 to t_3, inserting a spin echo before the spin lock and, to make these 3D experiments, changing the spin echo time to t_2 from the default t_e.

```
list[3D, HSQC, TOCSY] =
(pulseSeq[3D, HSQC, #, TOCSY] =
  pulseSeq[2D, HSQC, #, TOCSY] //
   pipe[insert[spinEcho, on[N15],
    before[spinLock]], t2 -> t3, te -> t2];
 drawPulseSequence[3D, HSQC, #, TOCSY])& /@ hsqcList
```

This produces the diagram for the basic 3D HSQC TOCSY (see Fig. 4.3), and the diagrams for the three variants in [7]. The 2D TOCSY HSQC prescriptions are converted to the 3D prescriptions in a similar way by

```
list[3D, TOCSY, HSQC] =
 adaptAssignmentOf[list[3D, HSQC, TOCSY],
  by[sequence[HSQC, #, TOCSY] ->
   sequence[TOCSY, HSQC, #]]]
=>
list[3D, TOCSY, HSQC] =
 (pulseSeq[3D, TOCSY, #, HSQC] =
   pulseSeq[2D, TOCSY, #, HSQC] //
    pipe[insert[spinEcho, on[N15],
     before[spinLock]], t2 -> t3, te -> t2];
  drawPulseSequence[3D, TOCSY, #, HSQC])& /@ hsqcList
```

This produces the diagram for the basic 3D TOCSY HSQC (see Fig. 4.4) and the diagrams for the three variants in [7].

5. Discussion

The production of diagrams is the most conspicuous feature of this paper. Looking to the future of increasingly automated design and execution of experiments, we believe that an even stronger role of APSEQ will be in mechanizing the construction and analysis of large numbers of pulse sequences systematically from concise input. This could well be part of a trend that is paralleled in other areas of science.

APSEQ provides a notation to represent concurrent events, that comprise successive stages which occur consecutively in time. The notation includes features that express analogy in the construction of pulse sequences. The notation suggests further features to support "natural" ways of expressing requirements. The design of the software allows several kinds of extension. In particular, we are working on an interface with Güntert's POMA package. A product operator analysis requires more information than is needed to draw pulse sequence diagrams. This will be put in by `insert` commands. We paraphrase his `Iz//`a_1 `//`a_2 ... expression into `Iz//pipe[`$a_1, a_2, \ldots$`]` (**C19** explains `pipe`).

Also, `focusOn[`$i_1, c_1, i_2, c_2, \ldots$`]`, where $i_1, i_2, \ldots$ specify items in a pulse sequence, and $c_1, c_2, \ldots$ are comments, will be allowed as a further argument of the `draw` command, to put explanatory text between arrows above individual stages of a pulse sequence. (This is in response to a referee's suggestion.)

The work in this paper was started in the hope of finding applications of mechanized analogy and adaptation at higher levels than discussed in Section 4. We offer two speculative suggestions for the future.

1. NMR experiments that cyclically make measurements using one or more pulse sequences, analyze the results to determine algorithmically how these should be changed to proceed with an investigation, and make the changes mechanically. This would be in the spirit of the laboratory automation that is developing in the field of proteomics [13].
2. Using analogy to choose pulse sequence by reference to the chemical moieties being sought, suspected or found.

Acknowledgements

We thank P. Güntert, N. Jacobsen, L. Mueller, C. Pacheco, A. Palmer and J.B. Saunders for comments and advice.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.jmr.2010.01.009.

[1] P. Güntert. Symbolic NMR product operator calculations, Int. J. Quant. Chem. 106 (2006) 344–350.
[2] A. Jerschow, MathNMR: spin and spatial tensor manipulation in MATHEMATICA. J. Magn. Reson. 176 (2005) 7–14.
[3] C.K. Anand, A.D. Bain, Z. Nie, Simulation of steady-state NMR of coupled systems using Liouville space and computer algebra methods. J. Magn. Reson. 189 (2007) 200–209.
[4] M.P. Barnett, The literature of NMR work that uses symbolic calculation. Available from <http://www.princeton.edu/~michaelb/nmr/literature>
[5] M.P. Barnett. Mathscape and molecular integrals, J. Symbolic Comput. 42 (2007) 265-289.
[6] M.P. Barnett. Reasoning in symbolic computation, Commun. Comput. Algebra 42 (2008) 1-17.
[7] Available from <http://*node*/apsExamples.pdf>, *node*=www.princeton.edu/~nmr/apseq.
[8] Available from <http://*node*/apsParameters.pdf>.
[9] Available from <http://*node*/apsImplementation.pdf>.
[10] Available from <http://*node*/apsDistribution.tar.gz>.
[11] M.H. Levitt. Symmetry in the design of NMR multi-pulse sequences, J. Chem. Phys. 128 (2008) 052205, 1–15.
[12] S. Wolfram. The Mathematica Book, second ed. Addison-Wesley, New York, 1991, or later editions.
[13] R.D. King, J. Rowland, S.G. Oliver, et al., The automation of science. Science, 324 (2009) 85–89.