The luamathalign package*

Marcel Krüger
tex@2krueger.de

May 4, 2022

1 The problem

In most cases, `amsmath` makes it simple to align multiple equations in a `align` environment. But sometimes, special requirements come up.

Maybe one of your alignment points is in an exponent, or in a radical? The first attempts for such alignments often fail. For example, assume that you want to align the following radicals like this (at the x^3 term):

$$\begin{aligned}&\sqrt{1-3x+3x^2+(x-1)^3}\\ =&\sqrt{1-3x+3x^2+x^3-3x^2+3x-1}\\ =&\sqrt{x^3}\end{aligned}$$

"Just adding `&` at the alignment points" doesn't work:

```
\begin{align*}
  \sqrt{1-3x+3x^2+(&x-1)^3}\\
  =\sqrt{1-3x+3x^2+&x^3-3x^2+3x-1}\\
  =\sqrt{&x^3}
\end{align*}
```

fails with

```
! Missing } inserted.
<inserted text>
}
l.73 \end{align*}
```

Another problem are nested alignments. Take this sample from anonymous on TEX – LATEX StackExchange: We want alignment like

$$\begin{aligned}aaaa &= 1 && \text{for } X\\ bbbb &= 1 && \text{for } Y\\ \left.\begin{aligned}c &= 1\\ d &= 12\end{aligned}\right\} && \text{for } Z\end{aligned}$$

but in

*This document corresponds to luamathalign v0.3, dated 2022-05-04.

```
\begin{align*}
aaaa &= 1  &&\text{for $X$} \\
bbbb &= 1  &&\text{for $Y$} \\
\left. \begin{aligned}
  c &= 1 \\
  d &= 12
\end{aligned} \right\}&&&\text{for $Z$}
\end{align*}
```

there is not obvious way to align the equal signs in the nested `aligned` with the outer signs.

2 The solution

luamathalign provides solutions for both problems under LuaLaTeX:

`\AlignHere`

The most important new macro is `\AlignHere`: It generates an alignment point like `&`, but it can be used almost everywhere.

So problems like our first example can be implemented by just using `\AlignHere` instead of `&`:

```
\begin{align*}
  \sqrt{1-3x+3x^2+(\AlignHere x-1)^3}\\
  =\sqrt{1-3x+3x^2+{\AlignHere x}^3-3x^2+3x-1}\\
  =\sqrt{\AlignHere x^3}
\end{align*}
```

$$\begin{aligned}
&\sqrt{1-3x+3x^2+(x-1)^3}\\
=&\sqrt{1-3x+3x^2+x^3-3x^2+3x-1}\\
=&\sqrt{x^3}
\end{aligned}$$

Sadly, this doesn't really help with the nested alignment problem: Even if we use `\AlignHere` in the `aligned` environment, the alignment points would be inserted in the inner and not in the outer alignment. For such cases, there is a variant which allows to specify at which level the alignment should happen:

\SetAlignmentPoint
\ExecuteAlignment

The primary command for this is \SetAlignmentPoint⟨*number*⟩. When called with a negative number it specifies the nesting level. For example when ⟨*number*⟩ is -1 it is the same as \AlignHere, while for -2 it is aligning one level higher and so on.

For example, our nested alignment above wanted to align the inner aligned and the outer align* at the same point, so \SetAlignmentPoint-2 is used directly next to a inner alignment point (here &, \AlignHere would work too). Then the \ExecuteAlignment has to appear in the context of the outer align*, so it can be written e.g. directly before the next & of the outer align*:

```
\begin{align*}
aaaa &= 1  &&\text{for $X$} \\
bbbb &= 1  &&\text{for $Y$} \\
\left. \begin{aligned}
  c \SetAlignmentPoint-2 &= 1 \\
  d &= 12
\end{aligned} \right\}&&\text{for $Z$}
\end{align*}
```

$$\begin{aligned} aaaa &= 1 && \text{for } X \\ bbbb &= 1 && \text{for } Y \\ \left.\begin{aligned} c &= 1 \\ d &= 12 \end{aligned}\right\} && \text{for } Z \end{aligned}$$

If you do not want to keep track of the right nesting level you can explicitly mark a level and refer to it. To do so, use a non-negative ⟨*number*⟩. When \SetAlignmentPoint is used with a non-negative ⟨*number*⟩ then \ExecuteAlignment⟨*number*⟩ must be executed afterwards with the same ⟨*number*⟩ at a point where adding a & would add a valid alignment point at the right level.

Our example above could therefore also be written as

```
\begin{align*}
aaaa &= 1  &&\text{for $X$} \\
bbbb &= 1  &&\text{for $Y$} \\
\left. \begin{aligned}
  c \SetAlignmentPoint0 &= 1 \\
  d &= 12
\end{aligned} \right\}\ExecuteAlignment0 &&\text{for $Z$}
\end{align*}
```

$$\begin{aligned} aaaa &= 1 && \text{for } X \\ bbbb &= 1 && \text{for } Y \\ \left.\begin{aligned} c &= 1 \\ d &= 12 \end{aligned}\right\} && \text{for } Z \end{aligned}$$

This variant is also useful when working with custom alignment environment not prepared to work with luamathalign. By default \SetAlignmentPoint⟨*number*⟩ with negative numbers (and therefore also \AlignHere) only work with amsmath's {align}, {aligned} and their variants. If you have another environment which also follows similar alignment rules then you can either restrict yourself to non-negative ⟨*number*⟩s in combination with \ExecuteAlignment or patch these environments similar to what luamathalign does for amsmath.

3 The implementation

3.1 Lua

```
local properties   = node.get_properties_table()
local luacmd       = require'luamathalign-luacmd'
local hlist        = node.id'hlist'
local vlist        = node.id'vlist'
local whatsit      = node.id'whatsit'
local glue         = node.id'glue'
local user_defined = node.subtype'user_defined'
local whatsit_id   = luatexbase.new_whatsit'mathalign'
local node_cmd     = token.command_id'node'
local ampersand    = token.new(38, 4)

local mmode do
  for k,v in next, tex.getmodevalues() do
    if v == 'math' then mmode = k end
  end
  assert(mmode)
end

-- We might want to add y later
local function is_marked(mark, list)
  for n in node.traverse(list) do
    local id = n.id
    if id == hlist or id == vlist then
      if is_marked(mark, n.head) then return true end
    elseif id == whatsit and n.subtype == user_defined
        and n.user_id == whatsit_id and n.value == mark then
      return true
    end
  end
  return false
end
local function assert_unmarked(mark, list, ...)
  local marked = is_marked(mark, list)
  if marked then
  tex.error("Multiple alignment marks", "I found multiple alignment marks \z
      of type " .. mark .. " in an alignment where I already had an \z
      alignment mark of that type. You should look at both of them and \z
      decide which one is right. I will continue with the first one for now.")
  end
  return ...
end
local measure do
  local vmeasure
  local function hmeasure(mark, list)
    local x, last = 0, list.head
    for n in node.traverse(last) do
      local id = n.id
      if id == hlist then
        local w, h, d = node.rangedimensions(list, last, n)
        x, last = x + w, n
```

```
      local dx = hmeasure(mark, n)
      if dx then return assert_unmarked(mark, n.next, dx + x) end
    elseif id == vlist then
      local w, h, d = node.rangedimensions(list, last, n)
      x, last = x + w, n
      local dx = vmeasure(mark, n)
      if dx then return assert_unmarked(mark, n.next, dx + x) end
    elseif id == whatsit and n.subtype == user_defined
        and n.user_id == whatsit_id and n.value == mark then
      local w, h, d = node.rangedimensions(list, last, n)
      local after
      list.head, after = node.remove(list.head, n)
      return assert_unmarked(mark, after, x + w)
    end
  end
end
function vmeasure(mark, list)
  for n in node.traverse(list.head) do
    local id = n.id
    if id == hlist then
      local dx = hmeasure(mark, n)
      if dx then return assert_unmarked(mark, n.next, dx + n.shift) end
    elseif id == vlist then
      local dx = vmeasure(mark, n)
      if dx then return assert_unmarked(mark, n.next, dx + n.shift) end
    elseif id == whatsit and n.subtype == user_defined
        and n.user_id == whatsit_id and n.value == mark then
      local after
      list.head, after = node.remove(list.head, n)
      return assert_unmarked(mark, after, 0)
    end
  end
end
function measure(mark, head)
  local x, last = 0, head
  for n in node.traverse(last) do
    local id = n.id
    if id == hlist then
      local w, h, d = node.dimensions(last, n)
      x, last = x + w, n
      local dx = hmeasure(mark, n)
      if dx then return assert_unmarked(mark, n.next, head, dx + x) end
    elseif id == vlist then
      local w, h, d = node.dimensions(last, n)
      x, last = x + w, n
      local dx = vmeasure(mark, n)
      if dx then return assert_unmarked(mark, n.next, head, dx + x) end
    elseif id == whatsit and n.subtype == user_defined
        and n.user_id == whatsit_id and n.value == mark then
      local w, h, d = node.dimensions(last, n)
      local after
      head, after = node.remove(head, n)
      return assert_unmarked(mark, after, head, x + w)
    end
```

```
    end
    return head
  end
end

local isolate do
  local visolate
  local function hisolate(list, offset)
    local x, last = 0, list.head
    local newhead, newtail = nil, nil
    local n = last
    while n do
      local id = n.id
      if id == hlist then
        local w, h, d = node.rangedimensions(list, last, n)
        x, last = x + w, n
        local inner_head, inner_tail, new_offset = hisolate(n, offset - x)
        if inner_head then
          if newhead then
            newtail.next, inner_head.prev = inner_head, newtail
          else
            newhead = inner_head
          end
          newtail = inner_tail
          offset = x + new_offset
        end
        n = n.next
      elseif id == vlist then
        local w, h, d = node.rangedimensions(list, last, n)
        x, last = x + w, n
        local inner_head, inner_tail, new_offset = visolate(n, offset - x)
        if inner_head then
          if newhead then
            newtail.next, inner_head.prev = inner_head, newtail
          else
            newhead = inner_head
          end
          newtail = inner_tail
          offset = x + new_offset
        end
        n = n.next
      elseif id == whatsit and n.subtype == user_defined
          and n.user_id == whatsit_id then
        local w, h, d = node.rangedimensions(list, last, n)
        x = x + w
        list.head, last = node.remove(list.head, n)
        if x ~= offset  then
          local k = node.new(glue)
          k.width, offset = x - offset, x
          newhead, newtail = node.insert_after(newhead, newtail, k)
        end
        newhead, newtail = node.insert_after(newhead, newtail, n)
        n = last
      else
```

```
      n = n.next
    end
  end
  return newhead, newtail, offset
end
function visolate(list, offset)
  local newhead, newtail = nil, nil
  local n = list.head
  while n do
    local id = n.id
    if id == hlist then
      if dx then return assert_unmarked(mark, n.next, dx + n.shift) end
      local inner_head, inner_tail, new_offset = hisolate(n, offset)
      if inner_head then
        if newhead then
          newtail.next, inner_head.prev = inner_head, newtail
        else
          newhead = inner_head
        end
        newtail = inner_tail
        offset = new_offset
      end
      n = n.next
    elseif id == vlist then
      if dx then return assert_unmarked(mark, n.next, dx + n.shift) end
      local inner_head, inner_tail, new_offset = visolate(n, offset)
      if inner_head then
        if newhead then
          newtail.next, inner_head.prev = inner_head, newtail
        else
          newhead = inner_head
        end
        newtail = inner_tail
        offset = new_offset
      end
      n = n.next
    elseif id == whatsit and n.subtype == user_defined
        and n.user_id == whatsit_id then
      local after
      list.head, after = node.remove(list.head, n)
      if 0 ~= offset  then
        local k = node.new(glue)
        k.width, offset = -offset, 0
        newhead, newtail = node.insert_after(newhead, newtail, k)
      end
      newhead, newtail = node.insert_after(newhead, newtail, n)
      n = last
    else
      n = n.next
    end
  end
  return newhead, newtail, offset
end
function isolate(head)
```

```
    local x, last = 0, head
    local newhead, newtail, offset = nil, nil, 0
    local n = last
    while n do
      local id = n.id
      if id == hlist then
        local w, h, d = node.dimensions(last, n)
        x, last = x + w, n
        local inner_head, inner_tail, new_offset = hisolate(n, offset - x)
        if inner_head then
          if newhead then
            newtail.next, inner_head.prev = inner_head, newtail
          else
            newhead = inner_head
          end
          newtail = inner_tail
          offset = x + new_offset
        end
        n = n.next
      elseif id == vlist then
        local w, h, d = node.dimensions(last, n)
        x, last = x + w, n
        local inner_head, inner_tail, new_offset = visolate(n, offset - x)
        if inner_head then
          if newhead then
            newtail.next, inner_head.prev = inner_head, newtail
          else
            newhead = inner_head
          end
          newtail = inner_tail
          offset = x + new_offset
        end
        n = n.next
      elseif id == whatsit and n.subtype == user_defined
          and n.user_id == whatsit_id then
        local w, h, d = node.dimensions(last, n)
        x = x + w
        head, last = node.remove(head, n)
        if x ~= offset  then
          local k = node.new(glue)
          k.width, offset = x - offset, x
          newhead, newtail = node.insert_after(newhead, newtail, k)
        end
        newhead, newtail = node.insert_after(newhead, newtail, n)
        n = last
      else
        n = n.next
      end
    end
    return head, newhead
  end
end

local function find_mmode_boundary()
```

```
  for i=tex.nest.ptr,0,-1 do
    local nest = tex.nest[i]
    if nest.mode ~= mmode and nest.mode ~= -mmode then
      return nest, i
    end
  end
end

luatexbase.add_to_callback('post_mlist_to_hlist_filter', function(n)
  local nest = find_mmode_boundary()
  local props = properties[nest.head]
  local alignment = props and props.luamathalign_alignment
  if alignment then
    props.luamathalign_alignment = nil
    local x
    n, x = measure(alignment.mark, n)
    local k = node.new'glue'
    local off = x - n.width
    k.width, alignment.afterkern.width = off, -off
    node.insert_after(n.head, nil, k)
    n.width = x
  end
  return n
end, 'luamathalign')

```

The glue node is referred to as a kern for historical reasons. A glue node is used since this interacts better with lua-ul.

```
local function get_kerntoken(newmark)
  local nest = find_mmode_boundary()
  local props = properties[nest.head]
  if not props then
    props = {}
    properties[nest.head] = props
  end
  if props.luamathalign_alignment then
    tex.error('Multiple alignment classes trying to control the same cell')
    return token.new(0, 0)
  else
    local afterkern = node.new'glue'
    props.luamathalign_alignment = {mark = newmark, afterkern = afterkern}
    return token.new(node.direct.todirect(afterkern), node_cmd)
  end
end

local function insert_whatsit(mark)
  local n = node.new(whatsit, user_defined)
  n.user_id, n.type, n.value = whatsit_id, string.byte'd', mark
  node.write(n)
end
luacmd("SetAlignmentPoint", function()
  local mark = token.scan_int()
  if mark < 0 then
    for i=tex.nest.ptr,0,-1 do
      local t = tex.nest[i].head
```

```
      local props = properties[t]
      if props and props.luamathalign_context ~= nil then
        mark = mark + 1
        if mark == 0 then
          props.luamathalign_context = true
          return insert_whatsit(-i)
        end
      end
    end
    tex.error('No compatible alignment environment found',
      'This either means that \\SetAlignmentPoint was used outside\n\z
      of an alignment or the used alignment is not setup for use with\n\z
      luamathalign. In the latter case you might want to look at\n\z
      non-negative alignment marks.')
  else
    return insert_whatsit(mark)
  end
end, "protected")

function handle_whatsit(mark)
  token.put_next(ampersand, get_kerntoken(mark))
end
luacmd("ExecuteAlignment", function()
  return handle_whatsit(token.scan_int())
end, "protected")

luacmd("LuaMathAlign@begin", function()
  local t = tex.nest.top.head
  local props = properties[t]
  if not props then
    props = {}
    properties[t] = props
  end
  props.luamathalign_context = false
end, "protected")
luacmd("LuaMathAlign@end@early", function()
  local t = tex.nest.top.head
  local props = properties[t]
  if props then
    if props.luamathalign_context == true then
      handle_whatsit(-tex.nest.ptr)
    end
    props.luamathalign_context = nil
  end
end, "protected")
local delayed
luacmd("LuaMathAlign@end", function()
  local t = tex.nest.top.head
  local props = properties[t]
  if props then
    if props.luamathalign_context == true then
      assert(not delayed)
      delayed = {get_kerntoken(-tex.nest.ptr), ampersand}
    end
```

```
    props.luamathalign_context = nil
  end
end, "protected")
luatexbase.add_to_callback("hpack_filter", function(head, groupcode)
  if delayed and groupcode == "align_set" then
-- HACK: token.put_next puts the tokens into the input stream after the cell
-- is fully read, before the next starts. This will act as if the content was
-- written as the first element of the next field.
    token.put_next(delayed)
    delayed = nil
  end
  return true
end, "luamathalign.delayed")

luacmd("LuaMathAlign@IsolateAlignmentPoints", function()
  local main = token.scan_int()
  if not token.scan_keyword 'into' then
    tex.error'Expected "into"'
  end
  local marks = token.scan_int()
  local head, newhead = isolate(tex.box[main])
  tex.box[marks] = node.direct.tonode(node.direct.hpack(
      newhead and node.direct.todirect(newhead) or 0))
end, "protected")
```

3.2 LaTeX

The actual LaTeX package just loads the Lua module and patches amsmath:

```
\RequirePackage{iftex}
\RequireLuaTeX
\directlua{require'luamathalign'}
\IfPackageLoadedTF{amsmath}{%
  \@firstofone
}{%
  \AddToHook{package/amsmath/after}
}
{%
  \def\align@preamble{%
     &\hfil
      \strut@
      \setboxz@h{\@lign$\m@th\displaystyle{%
          \LuaMathAlign@begin##\LuaMathAlign@end}$}%
      \ifmeasuring@\savefieldlength@\fi
      \set@field
      \tabskip\z@skip
     &\setboxz@h{\@lign$\m@th\displaystyle{{}##}$}%
      \ifmeasuring@\savefieldlength@\fi
      \set@field
      \hfil
      \tabskip\alignsep@
  }
  \renewcommand{\start@aligned}[2]{%
    \RIfM@\else
        \nonmatherr@{\begin{\@currenvir}}%
    \fi
```

```
    \savecolumn@ % Assumption: called inside a group
    \alignedspace@left
    \if #1t\vtop \else \if#1b \vbox \else \vcenter \fi \fi \bgroup
        \maxfields@#2\relax
        \ifnum\maxfields@>\m@ne
            \multiply\maxfields@\tw@
            \let\math@cr@@@\math@cr@@@alignedat
            \alignsep@\z@skip
        \else
            \let\math@cr@@@\math@cr@@@aligned
            \alignsep@\minalignsep
        \fi
        \Let@ \chardef\dspbrk@context\@ne
        \default@tag
        \spread@equation % no-op if already called
        \global\column@\z@
        \ialign\bgroup
           &\column@plus
            \hfil
            \strut@
            $\m@th\displaystyle{\LuaMathAlign@begin##\LuaMathAlign@end}$%
            \tabskip\z@skip
           &\column@plus
            $\m@th\displaystyle{{}##}$%
            \hfil
            \tabskip\alignsep@
            \crcr
  }
  \edef\math@cr@@@alignedat{\LuaMathAlign@end@early
    \unexpanded\expandafter{\math@cr@@@alignedat}}
  \edef\math@cr{\LuaMathAlign@end@early
    \unexpanded\expandafter{\math@cr}}
  \edef\endaligned{\LuaMathAlign@end@early
    \unexpanded\expandafter{\endaligned}}
}
\protected\def\AlignHere{\SetAlignmentPoint\m@ne}
\begingroup
  \def\patch@finph@nt\setbox\tw@\null{%
    \LuaMathAlign@IsolateAlignmentPoints\z@ into \tw@
  }%
\expanded{\endgroup%
\protected\def\noexpand\finph@nt{%
  \unexpanded\expandafter\expandafter\expandafter{%
    \expandafter\patch@finph@nt\finph@nt
  }%
}}
\ExplSyntaxOff
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

U

V

Z