ApplyPauliTransferMap

```
SetDirectory @ NotebookDirectory[];
Import["../Link/QuESTlink.m"];
```

Doc

```
? ApplyPauliTransferMap
```

Symbol

ApplyPauliTransferMap[pauliString, ptMap] returns the Pauli string produced by the given PTMap acting upon the given initial Pauli string.

ApplyPauliTransferMap[pauliString, circuit] automatically transforms the given circuit (composed of gates, channels, and PTMs, possibly intermixed) into PTMaps before applying them to the given Pauli string.

For improved performance, gate parameters should be kept symbolic (and optionally substituted thereafter) so that algebraic simplification can identify zero elements without interference by finite-precision numerical errors.

This method uses automatic caching to avoid needless re-computation of an operator's PTMap, agnostic to the targeted and controlled qubits, at the cost of additional memory usage. Caching behaviour can be controlled using option "CacheMaps":

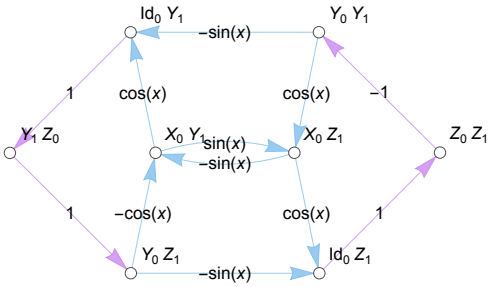
- "CacheMaps" -> "UntilCallEnd" (default) caches all computed PTMaps but clears the cache when ApplyPauliTransferMap[] returns.
- "CacheMaps" -> "Forever" maintains the cache even between multiple calls to ApplyPauliTransferMap[].
- "CacheMaps" -> "Never" disables caching (and clears the existing cache before computation), re-computing each operqtors' PTMap when encountered in the circuit.

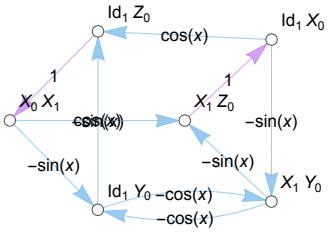
ApplyPauliTransferMap also accepts all options of CalcPauliTransferMap, like AssertValidChannels. See ?AssertValidChannels.

Correctness

Maps

```
map = CalcPauliTransferMap @ Circuit[H0 Rx0[x] C0[X1]];
DrawPauliTransferMap[map]
```






```
ApplyPauliTransferMap[X1 Z0, map]
```

X_0

```
ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0, map]
```

$c\, X_0 + b\, X_1 - a\, \text{Sin}[x]\, X_0\, Y_1 + a\, \text{Cos}[x]\, Z_1$

```
ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0, {map, map, map, map, map, map, map}]
```

$$-\frac{1}{2} c\, (1 + 3\, \text{Cos}[2\, x])\, \text{Sin}[x]^2\, X_0 + b\, X_1 + c\, \text{Cos}[x]^5\, X_0\, X_1 + c\, \text{Cos}[x]\, \text{Sin}[x]\, \left(\text{Cos}[x]^2 + \text{Cos}[x]^4 - 2\, \text{Sin}[x]^2\right)\, Y_0 + c\, \text{Cos}[x]^2\, (-1 + 2\, \text{Cos}[2\, x])\, \text{Sin}[x]\, X_1\, Y_0 + a\, \text{Sin}[x]\, \left(-\text{Cos}[x]^2 + \text{Sin}[x]^6 - \text{Sin}[2\, x]^2\right)\, X_0\, Y_1 - \frac{1}{8} a\, \text{Cos}[x]\, (3 - 12\, \text{Cos}[2\, x] + \text{Cos}[4\, x])\, Y_0\, Y_1 + \frac{1}{2} c\, \text{Sin}[x]\, \text{Sin}[4\, x]\, Z_0 + c\, \text{Cos}[x]^2\, \left(3 + \text{Cos}[x]^2\right)\, \text{Sin}[x]^2\, X_1\, Z_0 - \frac{1}{16} a\, (-1 - 16\, \text{Cos}[2\, x] + \text{Cos}[4\, x])\, \text{Sin}[2\, x]\, Y_1\, Z_0 - \frac{1}{8} a\, \text{Cos}[x]\, (3 - 20\, \text{Cos}[2\, x] + \text{Cos}[4\, x])\, \text{Sin}[x]^2\, Z_1$$

```
4 ^ 2
str = Z0 + Y1;
Table[
     Length @ ApplyPauliTransferMap[str, ConstantArray[map, reps]],
     {reps, 1, 10}]
```

16

{2, 3, 5, 8, 10, 10, 10, 10, 10, 10}

Circuits

Test 1

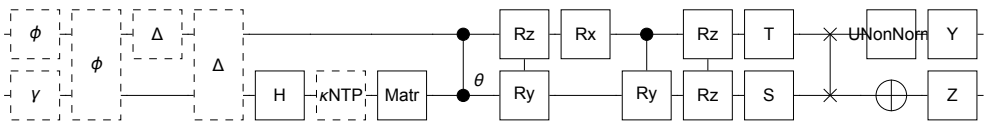
```
n = 4;
u = GetKnownCircuit["QFT", n];
m = CalcCircuitMatrix[u];

ρin = RandomComplex[{-1 - i, 1 + i}, {2^n, 2^n}];
ρout = m . ρin . ConjugateTranspose[m];

σin = GetPauliString[ρin];
σout = ApplyPauliTransferMap[σin, u];

CalcPauliExpressionMatrix[σout] - ρout // Abs // Max
Chop[%] === 0
```

6.66134×10^{-16}

True

Test 2

```
n = 2;
circ = Circuit[Damp0[a] Deph1[b] Deph0,1[c] Depol1[d]
     Depol1,0[e] H0 Id1 KrausNonTP0@RandomComplex[{0, i + 1}, {3, 2, 2}]
     Matr0@{{a, b}, {c, d}} C0@Ph1[h] R[i, Y0 Z1] Rx1[x] C1@Ry0[y]
     Rz0,1[z] S0 T1 SWAP0,1 UNonNorm 1 @ {{a, b}, {c, d}} X0 Y1 Z0];
DrawCircuit[%]
```




```
ρin = X0 + 2 Y0 + 3 Y1 + 4 Z1;
```

real/valid

```
subs = {a → .1, b → .2, c → .3, d → .4,
    e → .5, f → 6, g → 7, h → 8, i → 9, x → π, y → Exp[-π], z → -1};
m = CalcCircuitMatrix[circ /. subs];

ρin = RandomComplex[{-1 - ⅈ, 1 + ⅈ}, {2^n, 2^n}];
ρout = m . Flatten @ Transpose @ ρin;
ρout = Transpose @ ArrayReshape[ρout, {2^n, 2^n}];

σin = GetPauliString[ρin];
σout = ApplyPauliTransferMap[σin, circ /. subs];

CalcPauliExpressionMatrix[σout] - ρout // Abs // Max
Chop[%] === 0
```

2.35514×10^{-16}

```
True
```

complex/invalid

```
subs[[All, 2]] += 3 ⅈ;
m = CalcCircuitMatrix[circ /. subs, AssertValidChannels → False];

ρin = RandomComplex[{-1 - ⅈ, 1 + ⅈ}, {2^n, 2^n}];
ρout = m . Flatten @ Transpose @ ρin;
ρout = Transpose @ ArrayReshape[ρout, {2^n, 2^n}];

σin = GetPauliString[ρin];
σout = ApplyPauliTransferMap[σin, circ /. subs, AssertValidChannels → False];

err = (CalcPauliExpressionMatrix[σout] - ρout) / ρout // Abs // Max
Chop[%] === 0
```

2.00154×10^{-15}

```
True
```

Fully-mixing channels

```
ApplyPauliTransferMap[X0, Deph0[.4]]
ApplyPauliTransferMap[X0, Deph0[.5]]
ApplyPauliTransferMap[X0, Deph0[.6], AssertValidChannels → False]
```

$(0.2 + 0.\,ⅈ)\ X_0$

```
0
```

$(-0.2 + 0.\,ⅈ)\ X_0$

```
ApplyPauliTransferMap[X0, Deph0,1[3 / 4]]
ApplyPauliTransferMap[X0, Depol0[3 / 4]]
ApplyPauliTransferMap[X0, Depol0,1[15 / 16]]
```

```
0
```

```
0
```

```
0
```

```
ApplyPauliTransferMap[X0, Damp0[1]]
```

```
0
```

Options

Caching

```
circ = Table[C0[X1], 500];
Timing @
 ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0 + Y2, circ, "CacheMaps" → "Never"]
Timing @
 ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0 + Y2, circ, "CacheMaps" → "Forever"]
```

```
{4.4422, b X1 + Y2 + c X1 Z0 + a X0 Z1}
```

```
{0.72076, b X1 + Y2 + c X1 Z0 + a X0 Z1}
```

```
ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0 + Y2, circ, "CacheMaps" → "Forever"];
DownValues[QuEST`Private`obtainCachedPTMap] // First
```

```
ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0 + Y2, circ, "CacheMaps" → "Never"];
DownValues[QuEST`Private`obtainCachedPTMap] // First
```

```
ApplyPauliTransferMap[a X0 Z1 + b X1 + c X1 Z0 + Y2,
  circ, "CacheMaps" → "UntilCallEnd"];
DownValues[QuEST`Private`obtainCachedPTMap] // First
```

```
HoldPattern[QuEST`Private`obtainCachedPTMap[{C1[X0]}, CacheMaps → Forever]] :↦
 PTMap0,1[0 → {{0, 1}}, 1 → {{1, 1}}, 2 → {{14, 1}}, 3 → {{15, 1}},
  4 → {{5, 1}}, 5 → {{4, 1}}, 6 → {{11, 1}}, 7 → {{10, -1}},
  8 → {{9, 1}}, 9 → {{8, 1}}, 10 → {{7, -1}}, 11 → {{6, 1}},
  12 → {{12, 1}}, 13 → {{13, 1}}, 14 → {{2, 1}}, 15 → {{3, 1}}]
```

```
HoldPattern[QuEST`Private`obtainCachedPTMap[
   QuEST`Private`compGate_, QuEST`Private`opts___]] :↦
 (QuEST`Private`obtainCachedPTMap[QuEST`Private`compGate, QuEST`Private`opts] =
   CalcPauliTransferMap[QuEST`Private`compGate, Sequence @@
     FilterRules[{QuEST`Private`opts}, Options[CalcPauliTransferMap]]])
```

```
HoldPattern[QuEST`Private`obtainCachedPTMap[
   QuEST`Private`compGate_, QuEST`Private`opts___]] :↦
 (QuEST`Private`obtainCachedPTMap[QuEST`Private`compGate, QuEST`Private`opts] =
   CalcPauliTransferMap[QuEST`Private`compGate, Sequence @@
     FilterRules[{QuEST`Private`opts}, Options[CalcPauliTransferMap]]])
```

AssertValidChannels

```
ApplyPauliTransferMap[X0 Y1, Depol0,1[x]]
ApplyPauliTransferMap[X0 Y1, Depol0,1[x], AssertValidChannels → False]
```

$$\frac{1}{15}\,(15 - 16\,x)\,X_0\,Y_1$$

$$\frac{1}{15}\left(15\,\sqrt{1-x}\,\text{Conjugate}\left[\sqrt{1-x}\right] - \sqrt{x}\,\text{Conjugate}\left[\sqrt{x}\right]\right) X_0\,Y_1$$

Errors

```
ApplyPauliTransferMap[X0, Deph0[.6]]
```

ApplyPauliTransferMap : Could not pre-compute the Pauli transfer maps due to the below error:

CalcPauliTransferMatrix : One or more channels could not be asserted as completely positive and trace-preserving (CPTP) and ergo could not be simplified. Prevent this error with AssertValidChannels -> False.

```
$Failed
```

```
ApplyPauliTransferMap[X₀, {blob₀}]
```

ApplyPauliTransferMap : Could not pre−compute the Pauli transfer maps due to the below error:

CalcPauliTransferMatrix : Cannot obtain conjugate of unrecognised or unsupported operator: $blob_0$

```
$Failed
```

```
ApplyPauliTransferMap[X₀, X₀, "CacheMaps" → "Spaghetti"]
ApplyPauliTransferMap[X₀, {X₀}, "CacheMaps" → "Spaghetti"]
ApplyPauliTransferMap[X₀, PTM₀[x], "CacheMaps" → "Spaghetti"]
ApplyPauliTransferMap[X₀, {PTM₀[x]}, "CacheMaps" → "Spaghetti"]
ApplyPauliTransferMap[X₀, PTMap₀[x], "CacheMaps" → "Spaghetti"]
ApplyPauliTransferMap[X₀, {PTMap₀[x]}, "CacheMaps" → "Spaghetti"]
```

ApplyPauliTransferMap : Option "CacheMaps" must be one of "Forever", "UntilCallEnd" or "Never". See ?ApplyPauliTransferMap .

```
$Failed
```

ApplyPauliTransferMap : Option "CacheMaps" must be one of "Forever", "UntilCallEnd" or "Never". See ?ApplyPauliTransferMap .

```
$Failed
```

ApplyPauliTransferMap : Option "CacheMaps" must be one of "Forever", "UntilCallEnd" or "Never". See ?ApplyPauliTransferMap .

```
$Failed
```

ApplyPauliTransferMap : Option "CacheMaps" must be one of "Forever", "UntilCallEnd" or "Never". See ?ApplyPauliTransferMap .

```
$Failed
```

ApplyPauliTransferMap : Option "CacheMaps" must be one of "Forever", "UntilCallEnd" or "Never". See ?ApplyPauliTransferMap .

```
$Failed
```

ApplyPauliTransferMap : Option "CacheMaps" must be one of "Forever", "UntilCallEnd" or "Never". See ?ApplyPauliTransferMap .

```
$Failed
```

```
ApplyPauliTransferMap[X₀, {X₀}, "BadOption" → True]
```

OptionValue : Unknown option BadOption for {ApplyPauliTransferMap , CalcPauliTransferMap}.

```
$Failed
```

```
ApplyPauliTransferMap[X₀, {X₀},
 "CombineStates" → "Only valid for CalcPauliTransferEval"]
```

OptionValue : Unknown option CombineStates for {ApplyPauliTransferMap , CalcPauliTransferMap}.

```
$Failed
```

```
ApplyPauliTransferMap[a, {H0}]
ApplyPauliTransferMap[X-1, {H0}]
ApplyPauliTransferMap[X0 X0, {H0}]
ApplyPauliTransferMap[X0 Y0, {H0}]
```

ApplyPauliTransferMap : Invalid arguments. See ?ApplyPauliTransferMap

```
$Failed
```

ApplyPauliTransferMap : Invalid arguments. See ?ApplyPauliTransferMap

```
$Failed
```

ApplyPauliTransferMap : Invalid arguments. See ?ApplyPauliTransferMap

```
$Failed
```

ApplyPauliTransferMap : Invalid arguments. See ?ApplyPauliTransferMap

```
$Failed
```