Schnittstelle zwischen Algorithmen/GUI und dem Quantensimulationsframework (QSFW)

Die Schnittstelle zwischen dem Quantensimulationsframework (QSFW) basiert auf einer Abfolge von Anweisungen, die vom Anwender erstellt wird (bspw. jemandem, der einen Algorithmus damit umsetzen will oder sich per GUI etwas zusammenklicken will) und dann an das *QSFW* übergeben, dort verarbeitet und ausgeführt wird.

Die Abfolge der Anweisungen ist grundsätzlich in einer Datei zu hinterlegen, vorzugsweise mit der Endung `.qs`. Im Folgenden wird das Format der Anweisungsabfolge sowie der Anweisungen selbst beschrieben.

Fragen und Verbesserungsvorschläge sind an jonas.jelonek@hs-nordhausen.de zu richten.

Begrifflichkeiten

Das *QSFW* baut, wie anderen deutlich umfangreichere Frameworks auch, auf der Idee eines Quantenschaltkreises auf. Diese Idee hatten wir bereits in der Vorlesung mehrfach anklingen lassen und genutzt. Und es bietet natürlich einige Vorteile dass man einfach einen Schaltkreis und seine Eingangssignale/Qubits hat, in diesem Schaltkreis Gates platzieren und mit den Eingängen und Ausgängen verknüpfen kann. Wie in unseren Beispielen in der Vorlesung sind solche Aktionen wie `CNOT`, `Hadamard` oder `Messung` dann als Gate/Baustein wie in einem üblichen Schaltkreis zu sehen.

Andere Begriffe wie Anweisung, Funktion, Kommando, Argumente, imperativ usw., die verwendet werden, sollten für Informatiker bekannt sein.

Abfolge der Anweisungen (aka *QSFW* Programmcode)

Die Abfolge der Anweisungen kann als primitiver, imperativer Programmcode verstanden werden, mit dem dem *QSFW* gesagt wird, was gemacht werden soll. Eine Anweisung besteht immer aus den zwei Komponenten 'Funktion'/'Kommando' und 'Argumente' und hat stets ein festes Format:

```
<funktion/kommando>(<argumente>);
```

Der wachsame Leser wird merken, dass das von der Syntax her quasi ein Funktionsaufruf ist, wie man ihn aus den üblichen Programmiersprachen kennt. Diese Analogie wurde hier auch bewusst gewählt um es entsprechend einfach und verständlich zu halten.

`<funktion/kommando>` kann aus einer vorgegebenen, nicht erweiterbaren Liste von Funktionen/Kommandos gewählt werden. Diese Liste wird im nächsten Abschnitt besprochen. Ein(e) Funktion/Kommando macht immer irgendetwas, hier in unserem Fall des QSFW erstellt es bspw. einen Quantenschaltkreis/QuantumCircuit

Wie auch in `C` werden Anweisungen stets mit einem `;` (Semikolon) abgeschlossen, um Probleme, wie sie bspw. bei `Python` & co. auftreten, zu vermeiden. Man hat eine klare Trennung von Anweisungen und kann bspw. (auch wenn es das eventuell schlechter lesbar macht) mehrere Anweisungen in einer Zeile kombinieren. Auch wird keine spezielle Einrückung oder ähnliches benutzt.
Weiterhin sind Kommentare wie man sie aus `C` kennt möglich, also:

- einzeilige Kommentare beginnend mit `//` die den folgenden Text bis zum Zeilenumbruch als Kommentar kennzeichen
- mehrzeilige Kommentare beginnend mit `/*` und abschließend mit `*/`

Verfügbare Anweisungen

Grundlegende Funktionen

Funktion	Argumente	Beschreibung
`circuit`		`circuit` definiert einen Quantenschaltkreis, der für die nachfolgenden Aktionen die Grundlage bildet. Alle weiteren Funktionen können nur auf einem existierenden Quantenschaltkreis ausgeführt werden. Die Qubits, die mit diesem Befehl im Quantenschaltkreis initialisiert werden, bekommen eine ID zugewiesen die für nachfolgende Befehle genutzt wird und sich während der gesamten Lebenszeit des Schaltkreises nicht ändert. **Es gibt zwei Varianten des `circuit` Befehls die sich im Format der Argumente unterscheiden.**
(1)	`(n: int)`	Die Übergabe eines Integer-Werts, definiert den Schaltkreis mit `n` Qubits und jeweils dem Standardzustand von |0>. Die Qubits bekommen implizit eine aufsteigende ID zugewiesen, startend bei `0`.
(2)	`(q0: (str, int), q1: (str, int), ...)`	Werden dem Befehl mehrere komma-separierte 2-Tupel (syntaktisch: `(..,..)`) übergeben, können damit explizit die Qubits mit ihren IDs und den Anfangszuständen festgelegt werden. Beispielsweise würde ein Tupel `('a', 1)` festlegen, dass ein Qubit initialisiert wird mit der festen ID `a` und dem Anfangszustand `1`.

1-Qubit-Gates

Funktion	Argumente	Beschreibung
`ident`	`(qubit: str)`	Identitäts-Gate
`hadamard`	`(qubit: str)`	Hadamard-Gate
`phase`	`(qubit: str, angle: float)`	Phase-Shift-Gate; angle ist NICHT in **Grad** sondern **Radiant**, also basierend auf `π`. Als Winkel kann bspw. auch ein Ausdruck wie `2 * π` oder `π / 4` genutzt werden um nicht Dezimalzahlen nutzen zu müssen. Alternativ zum Unicode-Zeichen `π` kann auch `pi` genutzt werden.
`pauliX`	`(qubit: str)`	Pauli-X-Gate, auch NOT-Gate genannt.
`pauliY`	`(qubit: str)`	Pauli-Y-Gate
`pauliZ`	`(qubit: str)`	Pauli-Z-Gate; ist eine Spezialform des Phase-Shift-Gate mit einem Winkel von `π`.
`sphase`	`(qubit: str)`	S-Gate; ist eine Spezialform des Phase-Shift-Gate mit einem Winkel von `π/2`.
`tphase`	`(qubit: str)`	T-Gate; ist eine Spezialform des Phase-Shift-Gate mit einem Winkel von `π/4` (historisch auch `π/8`-Gate genannt).
`measure`	`(qubit: str)`	Führt eine Messung an einem Qubit durch. Die Messung erzeugt eine Ausgabe, deren Inhalt noch nicht genau festgelegt ist aber den gemessenen Zustand des Qubits widerspiegelt.

2-Qubit-Gates

Funktion	Argumente	Beschreibung
`cnot`	`(qubit0: str, qubit1: str)`	Controlled-NOT-Gate; Wie das Pauli-X-Gate nur kontrolliert durch `qubit0`.
`swap`	`(qubit0: str, qubit1: str)`	Swap-Gate; vertauscht die Zustände von `qubit0` und `qubit1`.
`cz`	`(qubit0: str, qubit1: str)`	Controlled-Z-Gate; Wie das Pauli-Z-Gate nur kontrolliert durch `qubit0`.
`cphase`	`(qubit0: str, qubit1: str, angle: float)`	Controlled-Phase-Shift-Gate; Wie das Phase-Shift-Gate nur kontrolliert durch `qubit0`.

3-Qubit-Gates

Funktion	Argumente	Beschreibung
`toffoli`	`(qubit0: str, qubit1: str, qubit2: str)`	Toffoli-Gate; auch Deutsch-Gate oder CCNOT-Gate genannt. Ist also wie ein NOT-Gate das durch `qubit0` und `qubit1` gesteuert `qubit2` negiert. Alternativ wie ein CNOT-Gate das durch ein weiteres Qubit gesteuert wird.
`cswap`	`(qubit0: str, qubit1: str, qubit2: str)`	Controlled-Swap-Gate; auch Fredkin-Gate genannt. Ist wie ein Swap-Gate und vertauscht die Zustände von `qubit1` und `qubit2` abhängig vom Zustand von `qubit0`.

Beispiel

Im Folgenden ein Beispiel für eine Anweisungsabfolge für das *QSFW*. **Das Beispiel dient nur dazu zu zeigen, wie ein passender Code für das *QSFW* geschrieben wird. Die Sinnhaftigkeit/Machbarkeit der Anweisungen und deren Abfolge sei erstmal dahingestellt.**

```
circuit(('a', 0), ('b', 1), ('c', 1));
ident('a');
pauliX('a');
pauliY('b');
pauliZ('c');
tphase('b');
phase('a', 2 * π);
measure('a');

cnot('a', 'b');
swap('b', 'c');
toffoli('a', 'b', 'c');
hadamard('b');
cnot('b', 'a');
measure('a');
```

Dieser *Code* sollte in einer Datei abgelegt werden die dann beim Aufruf des *QSFW* übergeben wird. Um das *QSFW* von der Kommandozeile aufzurufen, muss in den Hauptordner gewechselt werden, in dem sich auch die Ordner `doc` und `examples` befinden. Mit der genannten Datei kann das *QSFW* dann zur Verarbeitung wie folgt aufgerufen werden:

```
python3 -m qsfw /pfad/zur/programm/datei.qs
```

Alle Ausgaben und Ereignisse werden dann auf der Standardausgabe ausgegeben und können von der GUI verarbeitet oder direkt vom Anwender gesichtet werden.

Das *QSFW* bietet auch noch verschiedene Kommandozeilen-Parameter. Um alle möglichen Parameter mit einer Erklärung einzusehen, kann das *QSFW* wie folgt aufgerufen werden:

```
python3 -m qsfw -h
```

Um eins der vorhandenen Beispiele unter `examples` auszuführen, muss ebenfalls in der Kommandozeile in den Hauptordner gewechselt werden und dann das Beispiel wie folgt aufgerufen werden:

```
python3 examples/<beispiel>
```

Changelog

30.08.2023	Initiale Version
12.09.2023	• Die Typspezifikation für den Parameter `angle` der Funktionen `phase` und `cphase` wurde korrigiert. Er akzeptiert nur numerische Ausdrücke über den Typ `float` statt wie bisher auch `str`. Numerische Ausdrücke (bspw. '2 * e') sind ohne Anführungszeichen anzugeben. • Changelog hinzugefügt
15.10.2023	Die Informationen zur Nutzung bzw. wie man das QSFW aufruft wurden angepasst.