State Delta Resolution Algorithm

Erik Johnston

April 2018

1 Definitions

We first define some basic functions that correspond to the synapse code.

Let K be the set of all type/state key tuples and E the set of all events. We can then define arbitrary functions:

$$\begin{aligned} f &: F \to E \\ g &: G \to E \end{aligned} \tag{1}$$

which we call state maps, for $F, G \subset K$.

State maps have the additional properties that they are injective and that $\forall$ state maps f, g then $f(x) = g(y) \Rightarrow x = y$ (i.e. a given event can only be mapped to from a single state key).

We can compute the set of all "unconflicted state keys":

$$U_{f,g} = \{x \mid \forall x \in F \cap G,\ f(x) = g(x)\} \cup (F \triangle G) \tag{2}$$

i.e. the set of state keys where f and g don't conflict. Similarly, we define:

$$\begin{aligned} u_{f,g} : U_{f,g} &\longrightarrow E \\ x &\longmapsto \begin{cases} f(x), & \text{if } x \in F \\ g(x), & \text{otherwise} \end{cases} \end{aligned} \tag{3}$$

which gets the unconflicted event for a given state key.

We can also define a function on $C_{f,g} = (F \cup G) \setminus U_{f,g}$:

$$c_{f,g} : C_{f,g} \to E \tag{4}$$

which is used to resolve conflicts between f and g. Note that $c_{f,g}(x)$ is either $f(x)$ or $g(x)$.

Now we define:

$$r_{f,g} : F \cup G \longrightarrow E$$
$$x \longmapsto \begin{cases} u_{f,g}(x), & \text{if } x \in U_{f,g} \\ c_{f,g}(x), & \text{otherwise} \end{cases} \tag{5}$$

which we call the resolved state of f and g.

Note that this definition immediately implies that $\forall x \in U_{f,g}$ s.t. $g(x) = g'(x)$ then $r_{f,g}(x) = r_{f,g'}(x)$.

We define

$$\alpha : E \to \mathbb{P}(K) \tag{6}$$

to be the mapping of an event to the type/state keys needed to auth the event, and

$$\alpha_{f,g}(x) = (\alpha f(x)) \cup (\alpha g(x)) \tag{7}$$

which is the set of auth events required for $f(x)$ and $g(x)$. Note that $\alpha_{f,g}(x) \subset F \cup G$.

Further, we can define

$$\rho_{f,g}(x) = \bigcup_{n=0}^{\infty} (\alpha_{f,g})^n(x) \tag{8}$$

to be the auth chain of $f(x)$ and $g(x)$. This is well defined as there are a finite number of elements in $F \cup G$ and $\rho_{f,g} \to F \cup G$.

For examples of what α and ρ look like, see the appendix.

2 Results

If we consider the implementation of $c_{f,g}$ in Synapse we can see that it depends not only on f and g, but also on the resolved state of their auth events, i.e. $r_{f,g}\alpha_{f,g}$.

We can formalise this idea by defining the notion of dependency relations, which we do in the appendix. Using that notation, we can say that:

$$c_{f,g} \propto \{f,\ g,\ r_{f,g}\alpha_{f,g}\}$$

which leads to the following results.:

Lemma 1. $r_{f,g} \propto \{f\rho_{f,g}, g\rho_{f,g}\}$

Proof. First:

$$r_{f,g}\alpha_{f,g}(x) \ = (u_{f,g}\alpha_{f,g}(x)) \cup (c_{f,g}\alpha_{f,g}(x))$$

[1]but by definition $u_{f,g}\alpha_{f,g}(x) \propto \alpha_{f,g}(x)$, therefore:

$$r_{f,g}\alpha_{f,g} \propto \{f\alpha_{f,g},\ g\alpha_{f,g},\ c_{f,g}\alpha_{f,g}\}$$

so by induction, starting from $c_{f,g} \propto \{f,\ g,\ r_{f,g}\alpha_{f,g}\}$:

$$r_{f,g} \propto \{f(\alpha_{f,g})^i \mid \forall i \geq 0\} \cup \{g(\alpha_{f,g})^i \mid \forall i \geq 0\}$$

and the result follows since $f(\alpha_{f,g})^i \propto f\rho_{f,g}$

□

By inspecting the actual implementation of α we can define $\rho^{-1}_{f,g}(x)$ to be a function which $\forall y \in \rho_{f,g}(x)$ then $x \in \rho^{-1}_{f,g}(y)$. We can similarly define $\alpha^{-1}_{f,g}(x)$. Note that $\forall x,\ x \in \rho^{-1}_{f,g}(x)$

We now consider $g' : G' \to E$, where $g(x) = g'(x)$ except for $x \in G_\delta$, i.e. g' is a state map based on g.

Lemma 2. *For f, g, g' s.t. $\forall x \notin G_\delta,\ g(x) = g'(x)$, then $\forall x \notin \rho^{-1}_{f,g'}(G_\delta),\ r_{f,g}(x) = r_{f,g'}(x)$.*

[1]Note that e.g. $u_{f,g}\alpha_{f,g}(x)$ is shorthand for $\{u_{f,g}(y) \mid \forall y \in \alpha_{f,g}(x) \cap U_{f,g}\}$, i.e. we apply the function to the values in the given set that are also in the functions domain.

Proof. Let x be s.t. $r_{f,g}(x) \neq r_{f,g'}(x)$:

$$
\begin{aligned}
&\Rightarrow f\rho_{f,g}(x) \neq f\rho_{f,g'}(x) \text{ or } g\rho_{f,g}(x) \neq g'\rho_{f,g'}(x) \\
&\Rightarrow \exists y \in \rho_{f,g'}(x) \text{ s.t. } g(y) \neq g'(y) \\
&\Rightarrow y \in G_\delta \\
&\therefore \rho^{-1}_{f,g'}(y) \subseteq \rho^{-1}_{f,g'}(G_\delta) \\
&\Rightarrow x \in \rho^{-1}_{f,g'}(G_\delta)
\end{aligned}
$$

□

Corollary 3. *For f, g, g' s.t. $\forall x \notin G_\delta$, $g(x) = g'(x)$, then $\forall x \notin C_{f,g} \cap \rho^{-1}_{f,g'}(G_\delta)$, $r_{f,g}(x) = r_{f,g'}(x)$*

Proof. This follows from the previous result and that if $x \in U_{f,g} \setminus G_\delta$ then $r_{f,g}(x) = r_{f,g'}(x)$. □

This allows us to reuse most of the results of $r_{f,g}$ when calculating $r_{f,g'}$ if G_δ is small. In particular we can calculate the delta between the two functions without having to inspect $U_{f,g}$, which dramatically cuts down the amount of data used to compute deltas of resolved state of large state maps.

However, we can do better than this. We can note that $r_{f,g}(x)$ only depends on $r_{f,g}\alpha_{f,g}(x)$ for values of $\alpha_{f,g}(x)$ not in $U_{f,g}$. Concretely, this means for example that if G_δ includes the membership of the sender of a power level event, but the power level event is in $U_{f,g}$, then we don't need to recalculate all conflicted events—despite the membership event being in every event's auth chain.

Lemma 4. *$\forall x$ s.t. $r_{f,g}(x) \neq r_{f,g'}(x)$ then $\exists y_1, ..., y_n$ s.t. $y_n \in G_\delta$, $y_i \notin U_{f,g}$ and $y_{i+1} \in \alpha_{f,g'}(y_i)$*

Proof. If $r_{f,g}(x) \neq r_{f,g'}(x)$ then $\exists y \in G_\delta$ s.t. $y \in \rho_{f,g'}(x)$. By definition of $\rho_{f,g'}(x)$, $\exists y_0, ..., y_n$ s.t. $y_0 = x$ and $y_{i+1} \in \alpha_{f,g'}(y_i)$.

We know that $r_{f,g'}(x)$ depends on either $u_{f,g'}(x)$ or $c_{f,g'}(x)$, but if $x \in U_{f,g'}$ then there is no dependency on x's auth events and so $y_n = y_0 = x \in G_\delta$. Otherwise, we have $c_{f,g}(x) \neq c_{f,g'}(x)$, which depends on $f(x)$, $g'(x)$ or $r_{f,g'}(\alpha_{f,g'}(x))$. If $x \notin G_\delta$ then we know $f(x)$ and $g'(x)$ are the same, and so $r_{f,g}(\alpha_{f,g'}(x)) \neq r_{f,g'}(\alpha_{f,g'}(x)) \Rightarrow \exists y_1 \in \alpha_{f,g'}(x)$ s.t. $r_{f,g}(y_1) \neq r_{f,g'}(y_1)$.

Applying the above to y_1 then if $y_1 \in U_{f,g'} \Rightarrow y_1 = y_n \in G_\delta$. By induction $y_i \notin U_{f,g'}$ for $i < n$.

(Note that we can assume $y_i \notin G_\delta$ as otherwise we would pick $n = i$, and so if $y_i \notin U_{f,g} \Leftrightarrow y_i \notin U_{f,g'}$) □

We can use this approach and create an iterative algorithm for computing the set of state keys that need to be recalculated:

Algorithm 1 Calculate state keys needing to be recalculated

```
to_recalculate ← empty set of state keys
pending ← G_δ
while pending is not empty do
  x ← pop from pending
  if x ∉ U_{f,g} and x ∉ to_recalculate then
    add all in α^{-1}_{f,g'}(x) to pending
    add x to to_recalculate
  end if
end while
return to_recalculate
```

It should be noted that this only gives the set of keys that need to be recalculated, and not the full set that would be needed to actually recalculate them. The full set needs to include the auth events for each key, i.e.:

$$T \cup \bigcup_{x \in T} \alpha_{f,g}(x)$$

where T is the set *to_reacalculate*.

Appendices

A Dependency Relations

If we have functions a_f, b_f, etc we can define a notion of dependency. We say that a_f "depends only on" b_f, which is written as $a_f \propto b_f$, if

$$\forall x, g \text{ s.t. } a_f(x) \neq a_g(x) \implies b_f(x) \neq b_g(x)$$

This relation is transitive, i.e. if $a_f \propto b_f \propto c_f$ then $a_f \propto c_f$ as:

$$\begin{aligned} &\forall x, g \text{ s.t. } a_f(x) \neq a_g(x) \\ &\quad \implies b_f(x) \neq b_g(x) \\ &\quad \implies c_f(x) \neq c_g(x) \end{aligned}$$

Most functions depend on more than one other function, so we introduce the notation $a_f \propto \{b_f, c_f\}$ (a_f depends on the set of functions $\{b_f, c_f\}$) to mean

$$\begin{aligned} &\forall x, g \text{ s.t. } a_f(x) \neq a_g(x) \\ \implies &\text{ either } b_f(x) \neq b_g(x), \\ &\quad \text{or } c_f(x) \neq c_g(x) \end{aligned}$$

B Auth Functions

The following is the implementation of α in Synapse:

Listing 1: Definition of α

```
def auth_types_for_event(event):
    if event.type == EventTypes.Create:
        return []

    auth_types = []

    auth_types.append((EventTypes.PowerLevels, "", ))
    auth_types.append((EventTypes.Member, event.user_id, ))
    auth_types.append((EventTypes.Create, "", ))

    if event.type == EventTypes.Member:
        membership = event.content["membership"]
        if membership in [Membership.JOIN, Membership.INVITE]:
            auth_types.append((EventTypes.JoinRules, "", ))

        auth_types.append((EventTypes.Member, event.state_key, ))

        if membership == Membership.INVITE:
            if "third_party_invite" in event.content:
                key = (
                    EventTypes.ThirdPartyInvite,
                    event.content["third_party_invite"]["signed"]["token"]
                )
                auth_types.append(key)

    return auth_types
```

In particular the auth types are the same for all events except membership events. This means that $\alpha_{f,g}(x)$ is always the set { (“m.room.create”, “”), (“m.room.power_levels”, “”), (“m.room.member”, sender) } for non membership events.

If we have a room created by user u_1 (so the power levels, join rules etc. were all sent by them) and a state event sent by u_2, then the auth chain of that event (with state key x) is $\rho_{f,g}(x) = \{x$, (“m.room.create”, “”), (“m.room.power_levels”, “”), (“m.room.join_rules”, “”), (“m.room.member”, u_1), (“m.room.member”, u_2) }.

This gives a valid, though imperfect, possible definition of $\rho^{-1}_{f,g}$ where $\rho^{-1}_{f,g}(x) = \{x\}$ for all state tuples that aren't create/power_levels/membership/etc., and $\rho^{-1}_{f,g}(x) = F \cup G$ for those keys. This trivially satisfies the property that $\forall y \in \rho_{f,g}(x)$ then $x \in \rho^{-1}_{f,g}(y)$