Twitlang(er): interactions modeling language (and interpreter) for Twitter*

Rocco De Nicola[1], Alessandro Maggi[1], Marinella Petrocchi[2], Angelo Spognardi[2], and Francesco Tiezzi[3]

[1] IMT Institute for Advanced Studies, Lucca, Italy
{rocco.denicola,alessandro.maggi}@imtlucca.it

[2] CNR, Istituto di Informatica e Telematica, Pisa, Italy
{marinella.petrocchi,angelo.spognardi}@iit.cnr.it

[3] School of Science and Technology, University of Camerino, Italy
francesco.tiezzi@unicam.it

Abstract. Online social networks are widespread means to enact interactive collaboration among people by, e.g., planning events, diffusing information, and enabling discussions. Twitter provides one of the most illustrative example of how people can effectively interact without resorting to traditional communication media. For example, the platform has acted as a unique medium for reliable communication in emergency or for organising cooperative mass actions. This use of Twitter in a cooperative, possibly critical, setting calls for a more precise awareness of the dynamics regulating message spreading. To this aim, in this paper, we propose Twitlang, a formal language to model interactions among Twitter accounts. The operational semantics associated to the language allows users to clearly and precisely determine the effects of actions performed by Twitter accounts, such as post, retweet, reply-to or delete tweets. The language is implemented in the form of a Maude interpreter, Twitlanger, which takes a language term as an input and, automatically or interactively, explores the computations arising from the term. By relying on this interpreter, automatic verification of communication properties of Twitter accounts can be carried out via the analysis tools provided by the Maude framework. We illustrate the benefits of our executable formalisation by means of few simple, yet typical, examples of Twitter interactions, whose effects are somehow subtle.

Keywords: Social systems dynamics, Twitter, Formal semantics, Verification

1 Introduction

More than a personal microblogging site, Twitter has been transformed by common use to an information publishing venue. At August, 2014, stats reported 271 million of monthly active Twitter users, with an average of 500 million of tweets sent per day and

* Research supported by the European projects IP 257414 ASCENS and STReP 600708 QUANTICOL, the Italian PRIN 2010LHT4KM CINA, and the Registro.it project MIB (My Information Bubble).

about 307 tweets sent per user [1]. Popular public characters, such as actors and singers, as well as traditional mass media, such as radio, TV, and newspapers, currently use Twitter as a new media channel. Politicians commit notable part of their campaigns to their Twitter home pages, see, e.g., the last US presidential election event [2]. Naturally, the platform has raised the attention of the most famous brands, that massively use the site for business promotion [3]. Furthermore, it has been used for spreading alerts and activity information messages by civil protection departments and the most well-known humanitarian driving forces, e.g. [4].

One of the keys for the success of this socially-centric platform consists on its ease of use. Basically, Twitter users interact by posting *tweets*, textual messages up to 140 characters. Tweets can also carry pictures, URLs, or *mentions* to other users. Remarkably, mentions trigger *notifications* to the mentioned users. There are three types of possible relationships between Twitter users A and B: either A *follows* B, meaning that the tweets posted by B appear on A's Twitter timeline, or B follows A (with the complementary meaning), or both A and B follow each other. Of course, there is also the case of no relationship between A and B. Users may also *reply* to, or even *retweet*, any tweet, in order to spread to their followers what they think particularly worth of notice (leading to a capillary diffusion of tweets).

In the last recent years, researchers have focused their attention on several aspects of Twitter, from modeling the number and nature of follow relationships (see, e.g., [5]), to applying to tweets sentiment analysis and natural language processing techniques, in order to, e.g., discover trending topics and their correspondence to real events (see, e.g., [6]), to relying on machine learning for malicious accounts detection (e.g., [7]). In this paper, we focus on what probably represents one of the core aspects of the platform, that makes it so popular and widespread: the Twitter communication and interaction network. All those who like to use Twitter for socializing, being informed, interact within the community, must precisely know the dynamics of their tweets, say, e.g., which accounts are directly reachable by their tweets, or what happens if a tweet is deleted. A conscious usage of Twitter becomes even more crucial when it is used as a communication media to support (critical) collaborative work.

Despite the apparent easiness and simplicity of Twitter interactions, the achievement of a full user experience-awareness on Twitter should not be given for granted. Indeed, the effects of (a sequence of) Twitter interactions could be subtle. As simple examples, we invite the reader to consider the following three sequences of actions:

1. post a tweet t - reply to t - delete t;
2. post a tweet t - retweet t - undo the retweet;
3. post a tweet t - retweet t - retweet the retweet - delete t.

Without introducing here a formal notation, we give the intuition for such sequences. Sequences 1 and 2 involve two users, say $@mickey$ and $@goofy$, while sequence 3 involves also a third user, say $@donald$. In sequence 1, $@mickey$ posts a tweet t and $@goofy$ replies to that tweet, then $@mickey$ deletes t. In sequence 2, $@mickey$ posts a tweet t, $@goofy$ retweets t, and successively $@goofy$ cancels his retweet. In sequence 3, $@mickey$ posts a tweet t, $@goofy$ retweets t, then $@donald$ retweets $@goofy$'s retweet, and finally $@mickey$ deletes the original tweet t. The effects of the removal actions in these three interactions are quite different. In the first case, t is removed from any

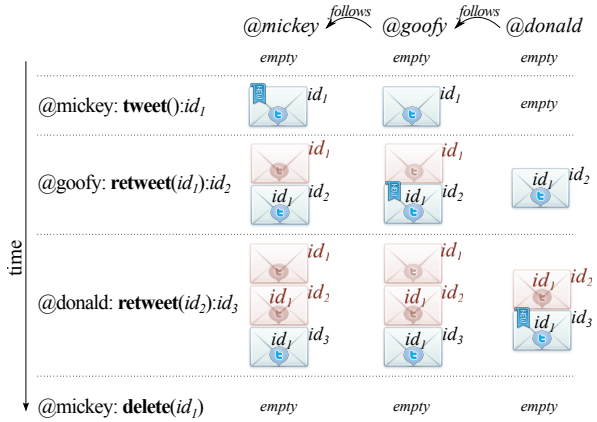

Fig. 1. Effects of an example Twitter interaction on users' accounts

timeline, while the reply still exists. In the second case, the fact that $@goofy$ cancels his retweet does not cause any effect to t, that still exists. Finally, in sequence 3, deleting t leads to the disappearance of the tweet and of all its retweets as well. Figure 1 gives a pictorial representation of sequence 3, from the point of view of the messages received by the three accounts under examination. For the sake of modeling, each tweet/retweet is labeled by a unique identifier id_j.

The previous interactions are just some of many example interactions users can engage on Twitter. Even these simple examples have effects that could be not fully intuitive for the community. In the following of the paper, we will show examples of interactions leading to more subtle and counterintuitive effects. This motivates the need for designing a rigorous model to trace, and hence analyse, Twitter interaction patterns.

In this paper, we propose a formalization of Twitter interactions, through Twitlang, a specification language describing a network of Twitter accounts and their behavior. The language has been inspired by process calculi (*à la* CCS [8]) and its semantics is defined in the SOS style [19] in terms of labeled transition systems. To the best of our knowledge, this is the first attempt to formally model the basic interactions resulting from users communicating on Twitter. The Twitlang formal semantics clearly determines the effects of the actions of a Twitter account, with respect to all the other accounts (including subtle and counterintuitive effects). This is determined "a priori", without the need of experimenting interactions and their effects case by case.

Besides being interesting per se, the Twitlang formal semantics has been implemented in the form of a Maude interpreter, called Twitlanger. It takes a language term, i.e. a specification of a network of Twitter accounts, as an input and performs an automatic or interactive exploration of the computations arising from the term. This also paves the way to automatic verification of communication properties of Twitter accounts (by using, e.g., the model checking facilities offered by the Maude toolset).

Road map The remainder of this paper is organized as follows. The next section presents the syntax and the semantics of Twitlang, focusing on specifying a simple Twitter interaction pattern. Then, we describe in Section 3 a sequence of Twitter interactions among three parties, which is peculiar for its counterintuitive visible outcome.

Table 1. Twitlang: syntax

(*Networks*)	$\mathcal{N}$	::=	$u : T : N : F : B \quad \mid \quad \mathcal{N}_1 \parallel \mathcal{N}_2$
(*Timelines*)	T	::=	$\epsilon \quad \mid \quad m \quad \mid \quad T_1, T_2$
(*Notification lists*)	N	::=	$\epsilon \quad \mid \quad m \quad \mid \quad N_1, N_2$
(*Messages*)	m	::=	$\langle id_{cur}, id_{ret}, id_{rep}, text, u_a, u_l, u_s \rangle$
(*Following lists*)	F	::=	$\epsilon \quad \mid \quad u \quad \mid \quad F_1, F_2$
(*Behaviours*)	B	::=	$\mathbf{nil} \quad \mid \quad a.B \quad \mid \quad B_1 + B_2 \quad \mid \quad B_1 \mid B_2 \quad \mid \quad K$
(*Actions*)	a	::=	$\mathbf{tweet}(text, x) \quad \mid \quad \mathbf{delete}(x)$
			$\mid \quad \mathbf{search}(\mathcal{P}, z)@t \quad \mid \quad \mathbf{retweet}(z, y) \quad \mid \quad \mathbf{undo}(y)$
			$\mid \quad \mathbf{reply}(z, text, U, x) \quad \mid \quad \mathbf{follow}(u) \quad \mid \quad \mathbf{unfollow}(u)$
(*Targets*)	t	::=	$u \quad \mid \quad \mathbf{all}$

We show that the semantics of the language is capable to precisely capture that subtle outcome, without the need for setting up empirical experiments. Section 4 describes the basic Maude modules of the Twitlanger interpreter. Section 5 is devoted to the related work in the area of Twitter modelling and analysis techniques. Finally, in Section 6 we discuss future work and conclude the paper. For the sake of readability, we have relegated to the Appendix the complete semantics of our formalism.

2 Twitlang: a formal language for modeling Twitter interactions

In this section, we introduce Twitlang, a formalism for modelling interactions among Twitter accounts. Specifically, we present both syntax and operational semantics of the language.

2.1 Syntax

The syntax of Twitlang is reported in Table 1.

A *network* $\mathcal{N}$ is a composition, by means of parallel operator $\parallel$, of *accounts* of the form $u : T : N : F : B$, where:

- u is a *username* that uniquely identifies the account;
- T is the *timeline*, i.e. the list of messages received from the account's followings or sent by the account;
- N is the list of *notifications* of the account, containing the messages where the account's username is mentioned and the replies to account's messages;
- F is the list of *followings* of the account;
- B is a model of the account's *behaviour*, expressed as a process performing Twitter actions.

A *message* is a data tuple of the form $\langle id_{cur}, id_{ret}, id_{rep}, text, u_a, u_l, u_s \rangle$, where:

- id_{cur} is the identifier of the (current) message;
- id_{ret} is the identifier of the original tweet the current message is a retweet of;
- id_{rep} is the identifier of the message the current message is a reply to;
- $text$ is the textual content of the message;
- u_a is the username of the author of the (retweeted or replied) original message;
- u_l is the username of the sender of the last retweet in a retweet chain;
- u_s is the username of the sender of the current message.

We will use the *null* symbol _ to leave unspecified a field of a message, as, for example, in the case of a new tweet, where the fields id_{ret}, id_{ret}, u_a and u_l are irrelevant. Moreover, we will exploit a *projection* function $m \downarrow_i$ that returns the i-th field of the message m. It is worth noticing that the identifiers used in a message act as *links* to other messages. Thus, given a message $\langle id_1, id_2, id_3, t, u_1, u_2, u_3 \rangle$, the identifier id_1 is a link to access all messages produced as replies to this message (i.e., the set of messages $\{m \mid m \downarrow_3 = id_1\}$), while the identifier id_3 can be used to access the previous message in the conversation (i.e., the message m such that $m \downarrow_1 = id_3$). Other messages can be iteratively retrieved from the already accessed ones. The navigation among messages via links can be done in Twitter by means of the functionalities *expand* and *view conversation*. As an example, let us consider the case of a reply to a reply of a tweet; the message m corresponding to the reply of the tweet permits accessing both the tweet message (by means of the id in the third field of m) and the second reply message (by means of the id in the first field of m).

Account *behaviours* are modelled by means of terms of a simple process algebra (actually, this is a simple variant of the well-known process algebra CCS [8], with specialised actions). Each process is built up from the *inert* process $\mathbf{nil}$ via *action prefixing* ($a.B$), *nondeterministic choice* ($B_1 + B_2$), *parallel composition* ($B_1 \mid B_2$), and *process invocation* (K). We assume that K ranges over a set of *process constants* that are used in (recursive) process definitions. We assume that each constant K has a single definition of the form $K \triangleq B$.

Processes can perform eight different kinds of *actions*. We use the following pairwise disjoint sets of variables: the set of *tweet variables* (ranged over by x), the set of *retweet variables* (ranged over by y), and the set of *message variables* (ranged over by z). We define three action prefixes $\mathbf{tweet}(text, x).B$, $\mathbf{retweet}(z, y).B$ and $\mathbf{reply}(z, text, U, x).B$ used to send messages to other accounts; they bind variables x and y in B. The receivers of such messages are determined according to follower-following relationships and presence of mentions in the content of messages, as formally described by the language semantics (described below). In particular, action $\mathbf{tweet}(text, x)$ produces a new tweet with content $text$, whose fresh message identifier is bound to the tweet variable x. Action $\mathbf{retweet}(z, y)$ permits retweeting a message identified by z; the fresh identifier of the retweet message is bound to the retweet variable y. Action $\mathbf{reply}(z, text, U, x)$ produces a message in response to the message identified by z; the produced message has content $text$, inherits all mentions from the replied message but for those specified in the set U of usernames[4], and its

[4] For the sake of simplicity, the set U is statically defined. This is adequate for the purpose of our study; a more dynamic definition of the set could be considered in further developments.

identifier is bound to variable x. Tweet and reply messages can be removed by means of action $\mathbf{delete}(x)$, while retweet messages by means of action $\mathbf{undo}(y)$. Actions $\mathbf{retweet}(z, y)$ and $\mathbf{reply}(z, text, U, x)$ act on a message that, at runtime, will replace the message variable z. This message is retrieved from the Twitter network by means of the (blocking) action $\mathbf{search}(\mathcal{P}, z)@t.B$, which indeed binds variable z in B. The action relies on a *predicate* $\mathcal{P}$ for selecting a message among those stored in a given account u (target $t = u$) or among all messages in the network (target $t = \mathbf{all}$). Predicates are boolean-valued expression obtained by logically combining the evaluation of (comparison) relations between message fields and values. An account can add or remove a username u to/from its following list F by means of actions $\mathbf{follow}(u)$ and $\mathbf{unfollow}(u)$, respectively.

We conclude the presentation of the syntax by showing how the examples shown in Figure 1 is rendered in our formalism.

Example 1 (Tweet-retweet-retweet-delete). Let us consider a network of three accounts with usernames u_m ($@mickey$), u_g ($@goofy$) and u_d ($@donald$), with empty timelines and notifications lists and such that u_g follows u_m and u_d follows u_g:

$$u_m : \epsilon : \epsilon : \epsilon : B_m \parallel u_g : \epsilon : \epsilon : u_m : B_g \parallel u_d : \epsilon : \epsilon : u_g : B_d$$

Account u_m posts a tweet, waits for a local message indicating that u_d has retweeted it, and then deletes it. Account u_g (resp. u_d) reads a local message from u_m (resp. u_g) and retweets it. This is rendered by the following behaviours:

$$\begin{aligned} B_m &= \mathbf{tweet}(Hello, x).\, \mathbf{search}(\downarrow_7 = u_d, z)@u_m.\, \mathbf{delete}(x).\, \mathbf{nil} \\ B_g &= \mathbf{search}(\downarrow_7 = u_m, z')@u_g.\, \mathbf{retweet}(z', y).\, \mathbf{nil} \\ B_d &= \mathbf{search}(\downarrow_7 = u_g, z'')@u_d.\, \mathbf{retweet}(z'', y').\, \mathbf{nil} \end{aligned}$$

Predicate $\downarrow_7 = u$ is verified by a message m if its sender (i.e., $m \downarrow_7$) is the username u.

2.2 A glimpse of the semantics

We present here an excerpt of the operational semantics of Twitlang. We refer to the Appendix for a more complete account.

The operational semantics is given in terms of a labeled transition relation, whose definition relies on an auxiliary relation on behaviors $B \xrightarrow{\alpha} B'$ meaning that "B can perform a transition labeled α and become B' in doing so". Intuitively, all actions give rise to a transition labeled by the corresponding label α. For example, the rules for actions **tweet**, **retweet** or **reply** are as follows:

$$\mathbf{tweet}(text, x).B \xrightarrow{\mathbf{tweet}(text,\, id)} B[id/x]$$

$$\mathbf{retweet}(m, y).B \xrightarrow{\mathbf{retweet}(m,\, id)} B[id/y]$$

$$\mathbf{reply}(m, text, U, x).B \xrightarrow{\mathbf{reply}(m, (m\downarrow_7 \cdot m\downarrow_5 \cdot mentions(m\downarrow_4)) \setminus U \cdot text, id)} B[id/x]$$

When one of the above actions is executed, a fresh message id is generated and used to replace the corresponding variable x or y via a *substitution*, i.e. a function $[v/k]$

Table 2. Twitlang: operational semantics (excerpt of rules at network level)

$$\frac{B \xrightarrow{\mathbf{tweet}(text,id)} B' \qquad id \notin \mathrm{ids}(T,N,B)}{u : T : N : F : B \xmapsto{\langle id,\_,\_,text,\_,\_,u\rangle} u : (T, \langle id, \_, \_, text, \_, \_, u\rangle) : N : F : B'}$$

$$\frac{B \xrightarrow{\mathbf{retweet}(m,id)} B' \qquad id \notin \mathrm{ids}(T,N,B) \qquad m\downarrow_7 \neq u}{u : T : N : F : B \xmapsto{\langle id,m\downarrow_{2/1},\_,m\downarrow_4,author(m),m\downarrow_7,u\rangle} u : (T, \langle id, m\downarrow_{2/1}, \_, m\downarrow_4, author(m), m\downarrow_7, u\rangle) : N : F : B'}$$

$$\frac{B \xrightarrow{\mathbf{reply}(m,text,id)} B' \qquad id \notin \mathrm{ids}(T,N,B)}{u : T : N : F : B \xmapsto{\langle id,\_,m\downarrow_1,text,m\downarrow_7,\_,u\rangle} u : (T, \langle id, \_, m\downarrow_1, text, m\downarrow_7, \_, u\rangle) : N : F : B'}$$

$$\frac{\mathcal{N} \xmapsto{m} \mathcal{N}' \qquad m\downarrow_1 \notin \mathrm{ids}(T,N,B)}{\mathcal{N} \parallel u : T : N : F : B \xmapsto{m} \mathcal{N}' \parallel u : (T \oplus^F m) : (N \oplus^u m) : F : B}$$

mapping variable k to value v. As clarified later, the freshness of message identifiers is ensured by operational rules at network level. The message text within the label produced by a **reply** action consists of a mention to the sender of message m, a mention to the author of the original tweet, all mentions included in the text of m (retrieved by means of the *mention retrieval* function $mentions(text)$) except those in U and, of course, the text of the reply (which may include new mentions).

The rules below state that the execution of an action permits to take a decision between alternative behaviors (left rule), while execution of parallel actions is interleaved (right rule):

$$\frac{B_1 \xrightarrow{\alpha} B_1'}{B_1 + B_2 \xrightarrow{\alpha} B_1'} \qquad \frac{B_1 \xrightarrow{\alpha} B_1'}{B_1 \mid B_2 \xrightarrow{\alpha} B_1' \mid B_2}$$

Now, the labeled transition relation on networks is given by the rules (an excerpt of which are) in Table 2. We write $\mathcal{N} \xmapsto{\lambda} \mathcal{N}'$ to mean that "$\mathcal{N}$ can perform a transition labeled λ and become $\mathcal{N}'$ in doing so". Transition labels are generated by the following production rule:

$$\lambda ::= m \mid \mathbf{delete}(id) \mid \mathbf{undo}(id) \mid u\colon \mathbf{found}(m) \mid u\colon \mathbf{added}(u') \mid u\colon \mathbf{removed}(u')$$

meaning that a message m has been transmitted, the tweet/reply identified by id and its related messages have been deleted, the retweet identified by id has been deleted, a message m is retrieved by u, the account u' has been added to the following list of u, the account u' has been removed from the following list of u, respectively.

The first rule shown in Table 2 transforms a **tweet** label into a network label m representing the message generated by the action. The message is inserted in the timeline of the account. Notably, premise $id \notin \mathrm{ids}(T,N,B)$ checks that the message id is fresh

in the considered account (in fact, function $\mathrm{ids}(\cdot)$ returns all identifiers used in the terms passed as arguments).

The second rule is similar; the extra premise $m \downarrow_7 \neq u$ permits blocking a retweet of a message generated by the same account u (indeed, this is not allowed in Twitter). Notice that this time the second field of the produced message records the identifier of the original tweet. If m is a retweet, this information is retrieved from the second field of m, while in case of tweet or reply it is retrieved from the first field. This is achieved by resorting to a particular *projection* function $m \downarrow_{i/j}$, which stands for $m \downarrow_i$ if $m \downarrow_i \neq \_$, otherwise $m \downarrow_j$. Similarly, the fifth field is determined by means of function $author(m)$ that returns $m \downarrow_5$ if $m \downarrow_2 \neq \_$ (i.e., m is a retweet), otherwise (i.e., m is a tweet or a reply) it returns $m \downarrow_7$. Moreover, the text of the retweet is the same of that of the retweeted message (indeed, in Twitter the **retweet** action does not allow to modify the text of the retweeted message).

The third rule of Table 2 is similar; the rule properly records identifier and author of the replied message m in the third and fifth field of the generated message, respectively.

The forth rule takes care of delivering a new message to all the accounts of the network that have to receive it. In particular, this rule should be repeatedly applied in order to consider one by one all the accounts. For each account is checked if the identifier of the message is fresh. In this way, at the end of the inference of the transition, the global freshness of the identifier is ensured. Notably, this does not require to use a restriction operator à la π-calculus [9], because the scope of the identifiers is always global, i.e. each user potentially can access every tweet in the network (in Twitter, for example, it is possible to access the messages sent and received by any user by visiting his/her Twitter page). The possible insertion of the message in the timeline and notification list of the considered account u is regulated by the following insertion operators:

- *tweet insertion* $T \oplus^F m$: a message m is inserted in the timeline T of an account only if the sender of m is in the following list F of this account;
- *notification insertion* $N \oplus^u m$: a message m is inserted in the notification list N of an account with username u only if u is mentioned in the text of m, or m is a retweet whose original tweet message has been sent by u, or m is a reply to a message sent by u.

Example 2 (Tweet-retweet-retweet-delete). Let $\mathcal{N}$ be the network defined in the Example 1. The behaviour B_m of the account u_m can evolve as follows:

$$B_m \xrightarrow{\mathbf{tweet}(Hello,id_1)} B'_m$$

Now, by applying the first rule in Table 2, the message $m_1 = \langle id_1, \_, \_, Hello, \_, \_, u_m \rangle$ is produced. Then, by applying the last rule in Table 2, m_1 is delivered to u_g (since u_g is a follower of u_m). Thus, the resulting transition is:

$$\mathcal{N} \xmapsto{\langle id_1,\_,\_,Hello,\_,\_,u_m\rangle} \mathcal{N}' = u_m : m_1 : \epsilon : \epsilon : B'_m \parallel u_g : m_1 : \epsilon : u_m : B_g \parallel u_d : \epsilon : \epsilon : u_g : B_d$$

Similarly, u_g and u_d perform their actions as follows:

$$\mathcal{N}' \xmapsto{u_g:\mathbf{found}(m_1)} \xmapsto{m_2} \xmapsto{u_d:\mathbf{found}(m_2)} \xmapsto{m_3} \mathcal{N}'' =$$
$$u_m : m_1 : (m_2, m_3) : \epsilon : B'_m \parallel u_g : (m_1, m_2) : m_3 : u_m : \mathbf{nil} \parallel u_d : (m_2, m_3) : \epsilon : u_g : \mathbf{nil}$$

where m_2 and m_3 are $\langle id_2, id_1, \_, Hello, u_m, u_m, u_g \rangle$ and $\langle id_3, id_1, \_, Hello, u_m, u_g, u_d \rangle$, respectively. Finally, u_m performs the search and delete actions:

$$\mathcal{N}'' \xrightarrow{u_m:\mathbf{found}(m_3)} \xrightarrow{\mathbf{delete}(id)} \mathcal{N}''' = u_m : \epsilon : \epsilon : \epsilon : \mathbf{nil} \parallel u_g : \epsilon : \epsilon : u_m : \mathbf{nil} \parallel u_d : \epsilon : \epsilon : u_g : \mathbf{nil}$$

As in Figure 1, the action produces a domino-effect that removes all messages from the timelines and notification lists.

3 An example interaction with counterintuitive effects

Twitter provides users with a basic set of simple features to communicate each other over the platform. Despite the apparent simplicity of such features, the combination of some communication actions can lead to counterintuitive effects.

We consider three Twitter accounts, say $@mickey$, $@donald$, and $@goofy$. We suppose that the three accounts belong to three distinct researchers, *Mickey Mouse*, *Donald Duck*, and *Goofy*, respectively. *Mickey* and *Donald* are colleagues and follow each others on Twitter, while *Goofy* is neither a follower nor a following of both. This scenario is rendered in our formalism as the following network (for the sake of presentation, we consider empty the timelines and notifications lists of the accounts at the beginning of the interaction):

$$u_m : \epsilon : \epsilon : u_d : B_m \;\parallel\; u_d : \epsilon : \epsilon : u_m : B_d \;\parallel\; u_g : \epsilon : \epsilon : \epsilon : B_g$$

Mickey is attending a conference on Social Informatics and listens with interest to *Goofy*'s talk on his recent results on using formal methods for the specification of the Twitter interaction patterns. Since *Mickey* and *Donald* are performing research on very related topics, *Mickey* sends an enthusiastic tweet mentioning both *Donald* and *Goofy*, with the following text: "$@donald$ *great work by* $@goofy$ *on #formalmethods and Twitter! Let's start a collaboration!*". Thus, the behavior of the *Mickey*'s account is:

$$B_m = \mathbf{tweet}(\text{“}u_d \textit{ great work by } u_g \textit{ on} \#formalmethods \textit{ and } Twitter! \ldots\text{”}, x).\, B'_m$$

Such a tweet, called hereafter the original tweet and denoted by m_1, appears 1) on *Donald*'s user timeline, since *Donald* follows *Mickey*, and on *Donald*'s notifications list, since *Donald* has been mentioned; 2) on *Goofy*'s notifications list, since *Goofy* has been mentioned, but *Goofy* does not follow *Mickey*; and 3) on *Mickey*'s user timeline:

$$u_m : m_1 : \epsilon : u_d : B'_m \parallel u_d : m_1 : m_1 : u_m : B_d \parallel u_g : \epsilon : m_1 : \epsilon : B_g$$

It happens that *Donald* has listened some rumors on *Goofy*'s professional reputation. Quite recklessly, he replies to the original tweet, although removing the mention to him: in that reply, called hereafter the replying tweet and denoted by m_2, *Donald* writes the following "$@mickey$ *don't go for it, waste of time*". Note that mention to $@mickey$ is automatically inserted in the replying tweet, being it a reply to the original tweet sent by *Mickey*. By default, the reply contains all the mentions included in the original tweet, thus, in this case, it automatically contains $@goofy$. However, *Donald* manually

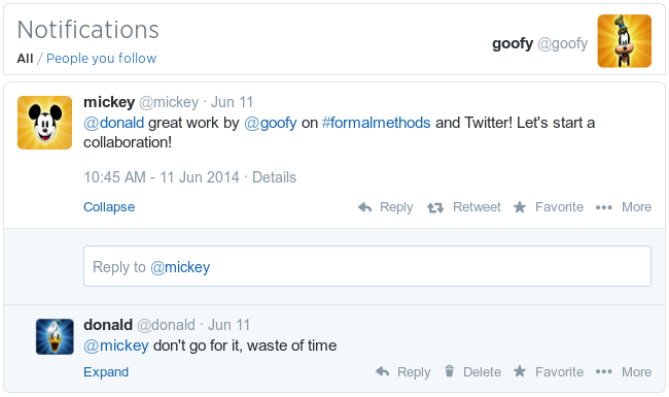

Fig. 2. Donald's reply is visible on Goofy's notification list

removes "@*goofy*" from the reply, before sending it. Thus, the behavior of the *Donald*'s account is:

$$B_d = \mathbf{search}(\downarrow_7 = u_m \wedge \#formalmethods \in hashtags(\downarrow_4), z)@u_d.\\ \mathbf{reply}(z, \text{“}u_m\ don't\ go\ for\ it,\ waste\ of\ time\text{”}, \{u_g\}, x').\, B'_d$$

Notably, the reply is triggered by the presence in the @*donald* account of a message whose sender is @*mickey* and whose text contains the hashtag #*formalmethods* (in fact, function $hashtags(\cdot)$ returns all hashtags in the text passed as argument).

Donald's reply 1) appears on *Mickey*'s user timeline, since *Mickey* follows *Donald*, and on *Mickey*'s notifications list, since *Mickey* has been mentioned; 2) appears on *Donald*'s user timeline; and 3) quite surprisingly, is added to a conversation on *Goofy*'s notifications list, even if the mention to *Goofy* has been removed. In particular, the reply is tied to the original tweet, and it is visible on *Goofy*'s notifications list upon clicking on the "expand" button. Figure 2 shows the screenshot of *Goofy*'s notifications list, upon clicking on the "expand" button. Formally, we have:

$$u_m : (m_1, m_2) : m_2 : u_d : B'_m \parallel u_d : (m_1, m_2) : m_1 : u_m : B'_d \parallel u_g : \epsilon : m_1 : \epsilon : B_g$$

where m_1 at u_g now allows *Goofy* accessing the message m_2. In fact, as explained in the section devoted to the presentation of our formalism, the identifiers in a message can be thought of as *links* to retrieve other messages. In our example, the identifier of m_1 (i.e., its first field) can be used to retrieve m_2, because $m_2 \downarrow_3$ is set to the m_1's identifier (since m_2 is a reply to m_1).

Finally, having seen the message of *Donald*, *Mickey* decides to remove his tweet, which is expressed in our formalism as an action $\mathbf{delete}(x)$. This removes all occurrences of m_1, leaving untouched those of m_2:

$$u_m : m_2 : m_2 : u_d : B''_m \parallel u_d : m_2 : \epsilon : u_m : B'_d \parallel u_g : \epsilon : \epsilon : \epsilon : B_g$$

Notice, even if the reply message is still around, *Goofy* now has no direct link to it.

4 Twitlanger: executable Twitlang in Maude

Maude is "a programming language that models (distributed) systems and the actions within those systems" [10]. The systems are specified by defining algebraic *data types* axiomatising systems' states, and *rewrite rules* axiomatising systems' local transitions.

In this section, we present Twitlanger, the interpreter for Twitlang written in Maude. Four basic Maude modules represent the core of Twitlanger: TWITLANG-SYNTAX, TWITLANG-CONTEXT, TWITLANG-SUPPORT and TWITLANG-SEMANTICS.

The functional module TWITLANG-SYNTAX provides declarations of sorts, e.g., networks, messages, actions and behaviours, and operators on those sorts that are defined in the language syntax. It also defines subsort relationships which are mainly used to capture the hierarchy between sets and respective elements. The module also provides reserved ground terms representing the names of actions (tweet, delete, search, etc.) and network-level labels (found, added, etc.). Given the similarities between behaviours in Twitlang and processes in CCS [8], we used Verdejo and Martí-Oliet state-of-the-art implementation of CCS in Maude [11] as a foundation for operators definition.

The functional module TWITLANG-CONTEXT defines the top-level behaviours' context that supports behaviour definition in terms of bindings to identifiers.

Module TWITLANG-SUPPORT defines equations that realise support operators used in rewrite rules for behaviour unfolding and network transitions.

Such rewrite rules are finally defined in TWITLANG-SEMANTICS, alongside additional operators and equations introduced to allow for a more compact and readable definition of the transition rules. The latter represent the operational semantics rules for behaviours and networks, an excerpt of which is given in Section 2.2 (while the complete set of rules is defined in Tables 4 and 5 in the Appendix).

Maude uses appropriate strategies for rules application. A Maude default strategy is implemented by the *rewrite* command, that explores one possible sequence of rewrites, starting by a set of rules and an initial state [10]. To prevent undesirable looping caused by recursive rewrites inside operators arguments, we have adopted an approach similar to the one described in [11]. Thus, in our implementation, the *rewrite* command can only be used to produce a one-step successor of a given state.

However, Maude provides another convenient command, *search*, which gives *a priori* all the possible sequences of rewrites between an initial and a final state supplied by the user. By providing a transitive closure to the network transitions, it is thus possible to use this command to evaluate arbitrarily long traces. Practically, since for certain recursively defined systems the search could not terminate, the command is decorated with an optional bound on the number of desired solutions and on the maximum depth of the search.

The example in Section 3 can be specified in the machine-readable syntax of Twitlang taken as input by Twitlanger. Then, the interpreter can be used to evaluate the evolution of the network, verifying that the exploration yields the expected outcome. Indeed, by issuing the following command[5]:

```
search example =>* T:Twitter .
```

[5] Both the command and its output use a shorthand notation - i.e. the terms `example`, `M1` and `M2` - that is equationally equivalent to a complex composition of terms.

we obtain a full unfolding of a rewrite trace

```
{M1}{Donald :Nfound(M1)}{M2}{Mickey :Nfound(M2)}{Mickey :Ndelete(1)}
```

up to the final state:

```
Donald : M2 : empty : Mickey : Bd'   ||   Goofy : empty : empty : none : Bg
||   Mickey : M2 : M2 : Donald : Bm''
```

Further analyses of the interactions can be performed by invoking *search* with the *such that* clause, effectively introducing a condition that the solutions have to fulfil. For instance, we may use the auxiliary operator *expand*, which evaluates accessible messages through direct linking (without resorting to the **search** action) from a specific user's perspective:

```
search example =>* T:Twitter such that ( M2 in expand(Goofy,1,T:Twitter) ) .
```

The command basically says "find all states of the system in which user *Goofy* can access message m_2 via a one-hop link". The output produced by the interpreter in this case is comprised of two solutions, the first one describing the trace and the state:

```
{M1}{Donald :Nfound(M1)}{M2}

Donald : ( M1 ; M2 ) : M1 : Mickey : Bd'   ||   Goofy : empty : M1 : none : Bg
||   Mickey : ( M2 ; M1 ) : M2 : Donald : ((search(predP7(Donald),z')@ Mickey) .
                                           delete(x) . Bm'')[1 / x])
```

which represents the system configuration after *Donald* replies to *Mickey*, meaning that indeed *Goofy* is able to easily access m_2 as soon as the message is published, even though it carries no mention of him. On the other hand, given that the only other solution found by the interpreter that satisfies the clause is the subsequent state in which *Mickey* has performed the **search** action, these results confirm that after deleting m_1 *Goofy* looses his only direct link to m_2 and, thus, he cannot access it without resorting to explicit **search**.

A more comprehensive overview of Twitlanger alongside the access to the complete Maude implementation of the Twitlanger modules and examples discussed in this paper, together with appropriate equations for all the declared operators, are available at `http://sysma.imtlucca.it/tools/twitlanger/`.

5 Related Work

To the best of our knowledge, there is no previously attempt to rigorously formalise Twitter interaction patterns. Instead, a series of blogs offer the general public some useful, yet informal, tips on tweets, retweets, and replies, see, e.g., [12].

Proposing a syntax and associated semantics describing the cause-effect relationships among communicating Twitter accounts should not be considered as a standalone work. Indeed, our formalisation aims at putting the rigorous basis for a uniform approach to Twitter accounts' properties specification and analysis. The first, yet significant, step in this direction is given by the implementation of the Twitlanger tool.

Interestingly, in the scientific literature there are several works on modelling and analysis of tweets' contents and their associated metadata. As an example, both work

in [13,14] exploit sentiment analysis techniques over real tweet-sets, to detect "public sentiment" and associate its fluctuations with a timeline of notable events that took place in the period tweets were collected. The authors of [15] address the problem of using text-mining tools to understand tweets (whose restricted length may prevent such tools from being employed to their full potential). The authors propose several schemes to train standard tools and compare their quality and effectiveness. In our work, instead, we mainly focus on analysing the interactions among users rather than on the content of their tweets.

Aiming at making tweets useful for recommendations, authors of [16] propose a method for enriching the semantics of tweets, by identifying and detailing, e.g., topics, persons, events mentioned in tweets. The usefulness of the platform for real-time crisis management has been tested by various work, see, e.g. [17,18], where technologies were investigated for understanding the semantic meaning of Twitter messages. Authors of [19] study the Twitter hashtags ability to represent real-world entities, by comparing hashtags characteristics with Semantic Web "strong identifiers" features. By analysing a dataset of Twitter conversations, work in [20] measures the "economy of attention" in the Twitter world. As predicted by Dunbar's theory, Twitter users can entertain a maximum of 100-200 stable relationships and are constrained by cognitive and biological constraints as well as in the real world. Authors of [5] provide a characterisation of the topological features of the Twitter follow graph, mainly aiming at answering questions related to the inner nature of the platform, e.g.: "Is Twitter a social network or an information network?". From the analysis they carried on, conclusions are that Twitter evolves towards a social network. Indeed, even if the "follow" relationships is primarily about information consumption, many relationships are instead "built on social ties". Similar issues are addressed in [21], where two Twitter networks are identified: a network made of followers and friends that shows a certain level of stability and a "topical" network, characterised by a high level of contingency. The work investigates how the two networks influence each other (for example, whether the participation in the same hashtag-based conversation changes the follower list of the involved accounts). Finally, work in [22] models information propagation through different social networks (among them, Twitter). Again, the above bunch of works concerns information and social aspects of Twitter, while we are interested in the effects of user interactions in terms of message spreading.

Remarkably, Twitter versatility and spread of use have made it the ideal arena for proliferation of anomalous accounts, that behave in unconventional ways. Literature has focused its attention on *spammers*, that is those accounts actively putting their efforts in spreading malware, sending spam, and advertising activities of doubtful legality (see, e.g., [7,23]) as well as on *fake followers*, corresponding to Twitter accounts specifically exploited to increase the number of followers of a target account, e.g., see [24]. Our research goal is to define an approach for distinguishing genuine accounts from anomalous one by making use of the analysis techniques enabled by the formal semantics and, in particular, by its Maude implementation.

To sum up, the above literature overview clearly highlights the research effort towards the characterisation of social dynamics inferred from Twitter studies and having an impact on real life (and vice versa). Our modeling approach, instead, focuses on

a novel study of Twitter interactions' effects from the point of view of Twitter users, with a special care on understanding the communication mechanisms underlying the message spreading. Besides this achievement, we think that our work can be extended in several directions in order to enable some of the analyses mentioned above. In fact, our formalism could serve as a uniform, common formal ground for modelling and analysing Twitter accounts' behaviour. For example, quantitative information could be added to model the frequency of actions (by resorting, e.g., to a stochastic approach).

6 Concluding remarks

We have presented Twitlang, a formal language to specify communication interactions on Twitter, from the point of view of the involved accounts. To the best of our knowledge, this is the first attempt to rigorously model communications on Twitter. By equipping the language with an operational semantics, it is possible to know in advance which are the effects of the basic actions that millions of Twitter users daily perform, without the need of setting up experiments (which, of course, we have extensively carried out to properly define our formal semantics). On top of the formal semantics, we have implemented Twitlanger, an interpreter of the language written in Maude.

It is worth noting that the language is currently able to capture the core aspects of Twitter communications, i.e., standard behavioural patterns, like, e.g., posting a tweet, replying to, or retweeting a particular tweet. However, it could be easily extended by giving both the syntax and the semantics rules for more specific features, as direct messages and blocking of an account. Concerning peculiar behaviours, an example, which perhaps not everyone is aware of, is as follows: putting a mention at the very beginning of a tweet implies that the tweet is sent only to the intersection of the author's followers and of the mentioned account's followers. This and other peculiarities, if considered relevant for specific analyses, could be dealt with in our approach.

As future work, we intend to incorporate in Twitlanger the Maude facilities supporting automatic analysis, thus enabling verification of Twitter interactions properties. Also, we aim at realising a user-friendly, on-line service, based on Twitlanger, through which the Twitter community can test what happens to their tweets, by means of simple questions and easy-to-understand answers.

References

1. Smith, C.: By The Numbers: 150+ Amazing Twitter Statistics. In: http://goo.gl/2Xr9X. (March 2015) Last checked March 21, 2015.
2. The Guardian: Barack Obama tweets the start to his 2012 re-election campaign. In: http://goo.gl/Uk6Av. (Apr 2011) Last checked March 21, 2015.
3. Brandwatch.com: Analysis of global brands' Twitter activity. In: http://goo.gl/C6MeU. (Dec 2012) Last checked March 21, 2015.
4. Save the Children: Hurricane Tips for Parents: How to Help Kids. In: http://goo.gl/vZynkt. (Jun 2014) Last checked March 21, 2015.
5. Myers, S.A., Sharma, A., Gupta, P., Lin, J.: Information Network or Social Network?: The Structure of the Twitter Follow Graph. In: WWW, ACM (2014) 493–498

6. Ritter, A., Cherry, C., Dolan, B.: Unsupervised modeling of twitter conversations. In: HLT-NAACL. (2010) 172–180
7. Stringhini, G., Kruegel, C., Vigna, G.: Detecting spammers on social networks. In: ACSAC, ACM (2010) 1–9
8. Milner, R.: Communication and concurrency. Prentice-Hall (1989)
9. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes. Inf. Comp. **100**(1) (1992) 1–77
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - a High-performance Logical Framework. Springer (2007)
11. Verdejo, A., Martí-Oliet, N.: Implementing CCS in Maude 2. In: WRLA. Volume 71 of ENTCS., Elsevier (2002) 239–257
12. Larson, D.: 9 Strange Things About Tweets, Retweets And DMs Every Twitter User Must Know. In: http://goo.gl/XyvAO. (Nov 2011) Last checked March 21, 2015.
13. Bollen, J., Mao, H., Pepe, A.: Modeling Public Mood and Emotion: Twitter Sentiment and Socio-Economic Phenomena. In: ICWSM. (2011)
14. Pak, A., Paroubek, P.: Twitter as a corpus for sentiment analysis and opinion mining. In: LREC, ELRA (2010)
15. Hong, L., Davison, B.D.: Empirical study of topic modeling in twitter. In: SOMA, ACM (2010) 80–88
16. Abel, F., Gao, Q., Houben, G.J., Tao, K.: Analyzing user modeling on twitter for personalized news recommendations. In: UMAP. (2011) 1–12
17. Abel, F., Hauff, C., Houben, G.J., Stronkman, R., Tao, K.: Twitcident: fighting fire with information from social web streams. In: WWW. (2012) 305–308
18. Mendoza, M., Poblete, B., Castillo, C.: Twitter Under Crisis: Can We Trust What We RT? In: SOMA, ACM (2010) 71–79
19. Laniado, D., Mika, P.: Making Sense of Twitter. In: ISWC. Volume 1. (2010) 470–485
20. Gonalves, B., Perra, N., Vespignani, A.: Modeling Users' Activity on Twitter Networks: Validation of Dunbar's Number. PLoS ONE **6**(8) (2011)
21. Rossi, L., Magnani, M.: Conversation practices and network structure in twitter. In: ICWSM. (2012)
22. Magnani, M., Rossi, L.: The ml-model for multi-layer social networks. In: ASONAM. (2011) 5–12
23. Yang, C., Harkreader, R., Gu, G.: Empirical evaluation and new design for fighting evolving twitter spammers. IEEE Information Forensics and Security **8**(8) (2013) 1280–1293
24. Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., Tesconi, M.: A Criticism to Society (as seen by Twitter analytics). In: DASec, IEEE (2014)
25. Plotkin, G.: A structural approach to operational semantics. J. Log. Algebr. Program. **60-61** (2004) 17–139

Appendix

In this Appendix we report the complete formal semantics of our Twitter formalism.

First, it is worth noticing that not all processes allowed by the syntax in Table 1 are semantically meaningful. Indeed, in a general term of the language, the messages stored in the accounts may not be consistent (e.g., we could have a message representing a retweet whose reference to the original tweet does not correspond to any tweet message). Thus, to ensure consistent terms we only consider *reachable* networks, i.e. networks obtained by means of reductions from networks with no stored messages.

Definition 1 (Reachable networks). *The set of reachable networks is the closure under* $\overset{\lambda}{\longmapsto}$ *(see below) of the set of terms generated by the following grammar:*

$$\mathcal{N} \ ::= \ u : \epsilon : \epsilon : F : B \quad | \quad \mathcal{N}_1 \parallel \mathcal{N}_2$$

The operational semantics is given in the SOS style [25] in terms of a structural congruence and of a labeled transition relation. Notably, the semantics is only defined for *closed* terms, i.e. terms without free variables. Indeed, we consider the binding of a variable as its declaration (and initialization), therefore free occurrences of variables at the outset in a term must be prevented since they are similar to uses of variables before their declaration in programs (which are considered as programming errors). Notice also that the semantics is defined over an *enriched syntax* that also includes those auxiliary terms resulting from replacing (free occurrences of) variables with the corresponding identifier.

The *structural congruence*, written $\equiv$, is defined as the smallest congruence relation on networks that includes the laws shown in Table 3.

$$B + \mathbf{nil} \equiv B \qquad B_1 + B_2 \equiv B_2 + B_1$$

$$(B_1 + B_2) + B_3 \ \equiv \ B_1 + (B_2 + B_3)$$

$$B \mid \mathbf{nil} \equiv B \qquad B_1 \mid B_2 \equiv B_2 \mid B_1$$

$$(B_1 \mid B_2) \mid B_3 \ \equiv \ B_1 \mid (B_2 \mid B_3)$$

$$\frac{K \triangleq B}{K \equiv B} \qquad \frac{B \equiv_\alpha B'}{B \equiv B'}$$

$$\frac{B \equiv B'}{u : T : N : F : B \equiv u : T : N : F : B'}$$

$$\mathcal{N}_1 \parallel \mathcal{N}_2 \equiv \mathcal{N}_2 \parallel \mathcal{N}_1 \qquad (\mathcal{N}_1 \parallel \mathcal{N}_2) \parallel \mathcal{N}_3 \equiv \mathcal{N}_1 \parallel (\mathcal{N}_2 \parallel \mathcal{N}_3)$$

Table 3. Twitter modeling language: structural congruence

Almost all laws are standard laws of process algebras: the first six laws are the monoid laws for $+$ and $\mid$ (i.e., it is associative and commutative, and has $\mathbf{nil}$ as identity el-

$$\mathbf{tweet}(text, x).B \xrightarrow{\mathbf{tweet}(text,id)} B[id/x] \quad \text{[B-TWEET]}$$

$$\mathbf{delete}(id).B \xrightarrow{\mathbf{delete}(id)} B \quad \text{[B-DELETE]}$$

$$\mathbf{search}(\mathcal{P}, z)@t.B \xrightarrow{\mathbf{search}(\mathcal{P},z)@t} B \quad \text{[B-SEARCH]}$$

$$\mathbf{retweet}(m, y).B \xrightarrow{\mathbf{retweet}(m,id)} B[id/y] \quad \text{[B-RETWEET]}$$

$$\mathbf{undo}(id).B \xrightarrow{\mathbf{undo}(id)} B \quad \text{[B-UNDO]}$$

$$\mathbf{reply}(m, text, U, x).B \xrightarrow{\mathbf{reply}(m,(m\downarrow_7 \cdot m\downarrow_5 \cdot mentions(m\downarrow_4))\setminus U \cdot text,id)} B[id/x] \quad \text{[B-REPLY]}$$

$$\mathbf{follow}(u).B \xrightarrow{\mathbf{follow}(u)} B \quad \text{[B-FOLLOW]}$$

$$\mathbf{unfollow}(u).B \xrightarrow{\mathbf{unfollow}(u)} B \quad \text{[B-UNFOLLOW]}$$

$$\frac{B_1 \xrightarrow{\alpha} B_1'}{B_1 + B_2 \xrightarrow{\alpha} B_1'} \text{[B-CHOICE]} \qquad \frac{B_1 \xrightarrow{\alpha} B_1'}{B_1 \mid B_2 \xrightarrow{\alpha} B_1' \mid B_2} \text{[B-PAR]}$$

Table 4. Twitter modeling language: operational semantics (behaviors)

ement); the seventh law permits unfolding a recursion; the eighth law equates alpha-equivalent behaviors, i.e. behaviors only differing in the identity of bound variables (alpha-equivalence is denoted by $\equiv_\alpha$); the ninth law permits lifting the structural congruence from behaviors to nets; the last two laws state that $\|$ is associative and commutative.

To define the labeled transition relation, we rely on an auxiliary relation on behaviors, which is defined as the smallest relation on behaviors generated by the rules in Table 4. We write $B \xrightarrow{\alpha} B'$ to mean that "B can perform a transition labeled α and become B' in doing so". Transition labels are generated by the following production rule

$$\begin{aligned} \alpha ::= \ & \mathbf{tweet}(text, id) \mid \mathbf{delete}(id) \mid \mathbf{search}(\mathcal{P}, z)@t \\ \mid \ & \mathbf{retweet}(m, id) \mid \mathbf{undo}(id) \mid \mathbf{reply}(m, text, id) \\ \mid \ & \mathbf{follow}(u) \mid \mathbf{unfollow}(u) \end{aligned}$$

Basically, all actions give rise to the corresponding label. When a **tweet**, **retweet** or **reply** is executed, a fresh message id is generated and used to replace the corresponding variable x or y via a *substitution*, i.e. a function $[v/k]$ mapping variable k to value v. Application of a substitution to a behavior, written $B[v/k]$, has the effect of replacing every free occurrence of k in B with v. As clarified later, the freshness of identifiers is ensured by operational rules at network level.

The message text within the label produced by a **reply** action is extended with the mentions inherited from the replied message m, except for those indicated in U. To this aim, we exploit a *mention retrieval* function $mentions(text)$ and the *removal* operator $text\backslash U$: the former returns the set of usernames mentioned in $text$, while the latter removes from $text$ all mentions to accounts belonging to the set U. Thus, in the label generated by the **reply** action, the text of the message consists of a mention to the sender of message m, a mention to the author of the original tweet, all mentions included in the text of m except those in U and, of course, the text of the reply. They are composed by means of the *concatenation* operator $\cdot$. Notably, new mentions can be added by means of the text of the reply.

Execution of an action permits to take a decision between alternative behaviors (rule [B-CHOICE]), while execution of parallel actions is interleaved (rule [B-PAR]).

The *labeled transition relation* is the smallest relation on closed networks generated by the rules in Table 5. We write $\mathcal{N} \overset{\lambda}{\longmapsto} \mathcal{N}'$ to mean that "$\mathcal{N}$ can perform a transition labeled λ and become $\mathcal{N}'$ in doing so". Transition labels are generated by the following production rule

$$\begin{array}{rcl} \lambda & ::= & m \quad | \quad \mathbf{delete}(id) \quad | \quad \mathbf{undo}(id) \quad | \quad u : \mathbf{found}(m) \\ & | & u : \mathbf{added}(u') \quad | \quad u : \mathbf{removed}(u') \end{array}$$

meaning that a message m has been transmitted, the tweet/reply identified by id and its related messages have been deleted, the retweet identified by id has been deleted, a message m is retrieved, the account u' has been added to the following list of u, the account u' has been removed from the following list of u, respectively.

Rule [N-TWEET] transforms a **tweet** label into a network label m representing the message generated by the action. The message is inserted in the timeline of the account. Notably, premise $id \notin \mathrm{ids}(T, N, B)$ checks that the message id is fresh in the considered account (in fact, function $\mathrm{ids}(\cdot)$ returns all identifiers used in the terms passed as arguments).

Rule [N-RETWEET] is similar; the extra premise $m \downarrow_7 \neq u$ permits blocking a retweet of a message generated by the same account u (indeed, this is not allowed in Twitter). Notice that this time the second field of the produced message records the identifier of the original tweet. If m is a retweet, this information is retrieved from the second field of m, while in case of tweet or reply it is retrieved from the first field. This is achieved by resorting to a particular *projection* function $m \downarrow_{i/j}$, which stands for $m \downarrow_i$ if $m \downarrow_i \neq \_$, otherwise $m \downarrow_j$. Similarly, the fifth field is determined by means of function $author(m)$ that returns $m \downarrow_5$ if $m \downarrow_2 \neq \_$ (i.e., m is a retweet), otherwise (i.e., m is a tweet or a reply) it returns $m \downarrow_7$. Moreover, the text of the retweet is the same of that of the retweeted message (indeed, in Twitter the **retweet** action does not allow to modify the text of the retweeted message).

Rule [N-REPLY] is similar; the rule properly records the identifier and author of the replied message m in the third and fifth field of the generated message, respectively.

Rule [N-DELIVER] takes care of delivering a new message to all account of the network that have to receive it. In particular this rule should be repeatedly applied for considering once at a time all accounts. For each account is checked if the identifier of the message is fresh. In this way, at the end of the inference of the transition, the

$$\frac{B \xrightarrow{\mathbf{tweet}(text,id)} B' \qquad id \notin \mathrm{ids}(T,N,B)}{u:T:N:F:B \xmapsto{\langle id,\_,\_,text,\_,\_,u\rangle} u:(T,\langle id,\_,\_,text,\_,\_,u\rangle):N:F:B'} \text{[N-TWEET]}$$

$$\frac{B \xrightarrow{\mathbf{retweet}(m,id)} B' \qquad id \notin \mathrm{ids}(T,N,B) \qquad m\downarrow_7 \neq u}{\begin{array}{c}u:T:N:F:B \xmapsto{\langle id,m\downarrow_{2/1},\_,m\downarrow_4,author(m),m\downarrow_7,u\rangle} \\ u:(T,\langle id,m\downarrow_{2/1},\_,m\downarrow_4,author(m),m\downarrow_7,u\rangle):N:F:B'\end{array}} \text{[N-RETWEET]}$$

$$\frac{B \xrightarrow{\mathbf{reply}(m,text,id)} B' \qquad id \notin \mathrm{ids}(T,N,B)}{u:T:N:F:B \xmapsto{\langle id,\_,m\downarrow_1,text,m\downarrow_7,\_,u\rangle} u:(T,\langle id,\_,m\downarrow_1,text,m\downarrow_7,\_,u\rangle):N:F:B'} \text{[N-REPLY]}$$

$$\frac{\mathcal{N} \xmapsto{m} \mathcal{N}' \qquad m\downarrow_1 \notin \mathrm{ids}(T,N,B)}{\mathcal{N} \parallel u:T:N:F:B \xmapsto{m} \mathcal{N}' \parallel u:(T \oplus^F m):(N \oplus^u m):F:B} \text{[N-DELIVER]}$$

$$\frac{B \xrightarrow{\mathbf{delete}(id)} B'}{u:T:N:F:B \xmapsto{\mathbf{delete}(id)} u:(T \ominus id):(N \ominus id):F:B' \not\downarrow_{id}} \text{[N-DELETE]}$$

$$\frac{\mathcal{N} \xmapsto{\mathbf{delete}(id)} \mathcal{N}'}{\mathcal{N} \parallel u:T:N:F:B \xmapsto{\mathbf{delete}(id)} \mathcal{N}' \parallel u:(T \ominus id):(N \ominus id):F:B \not\downarrow_{id}} \text{[N-DELPROPAG]}$$

$$\frac{B \xrightarrow{\mathbf{undo}(id)} B'}{u:T:N:F:B \xmapsto{\mathbf{undo}(id)} u:(T \oplus id):(N \oplus id):F:B' \not\downarrow_{id}} \text{[N-UNDO]}$$

$$\frac{\mathcal{N} \xmapsto{\mathbf{undo}(id)} \mathcal{N}'}{\mathcal{N} \parallel u:T:N:F:B \xmapsto{\mathbf{undo}(id)} \mathcal{N}' \parallel u:(T \oplus id):(N \oplus id):F:B \not\downarrow_{id}} \text{[N-UNDOPROPAG]}$$

$$\frac{B \xrightarrow{\mathbf{search}(\mathcal{P},z)@u} B' \qquad \exists\, m \in (T \cup N) \;:\; \mathcal{P}(m) = true}{u:T:N:F:B \xmapsto{u:\mathbf{found}(m)} u:T:N:F:B'[m/z]} \text{[N-SEARCH-U]}$$

$$\frac{B \xrightarrow{\mathbf{search}(\mathcal{P},z)@t} B'' \qquad (t = u' \;\vee\; t = \mathbf{all}) \qquad \exists\, m \in T' \;:\; \mathcal{P}(m) = true}{\begin{array}{c}u:T:N:F:B \parallel u':T':N':F':B' \xmapsto{u:\mathbf{found}(m)} \\ u:T:N:F:B''[m/z] \parallel u':T':N':F':B'\end{array}} \text{[N-SEARCH-T]}$$

$$\frac{B \xrightarrow{\mathbf{follow}(u')} B'' \qquad u' \notin F}{\begin{array}{c}u:T:N:F:B \parallel u':T':N':F':B' \xmapsto{u:\mathbf{added}(u')} \\ u:(T,\{m \in T' \mid m\downarrow_7 = u'\}):N:(F,u'):B'' \parallel u':T':N':F':B'\end{array}} \text{[N-FOLLOW1]}$$

$$\frac{B \xrightarrow{\mathbf{follow}(u')} B'' \qquad u' \in F}{\begin{array}{c}u:T:N:F:B \parallel u':T':N':F':B' \xmapsto{u:\mathbf{added}(u')} \\ u:T:N:F:B \parallel u':T':N':F':B'\end{array}} \text{[N-FOLLOW2]}$$

$$\frac{B \xrightarrow{\mathbf{unfollow}(u')} B''}{\begin{array}{c}u:T:N:F:B \parallel u':T':N':F':B' \xmapsto{u:\mathbf{removed}(u')} \\ u:(T\backslash\{m \in T \mid m\downarrow_7 = u'\}):N:(F\backslash u'):B'' \parallel u':T':N':F':B'\end{array}} \text{[N-UNFOLLOW]}$$

$$\frac{\mathcal{N}_1 \xmapsto{\lambda} \mathcal{N}_1' \qquad \lambda \in \{u:\mathbf{found}(m), u:\mathbf{added}(u'), u:\mathbf{removed}(u')\}}{\mathcal{N}_1 \parallel \mathcal{N}_2 \xmapsto{\lambda} \mathcal{N}_1' \parallel \mathcal{N}_2} \text{[N-PAR]}$$

$$\frac{\mathcal{N} \equiv \mathcal{N}_1 \xmapsto{\lambda} \mathcal{N}_2 \equiv \mathcal{N}'}{\mathcal{N} \xmapsto{\lambda} \mathcal{N}'} \text{[N-STR]}$$

Table 5. Twitter modeling language: operational semantics (networks)

global freshness of the identifier is ensured. Notably, this does not require to use a restriction operator à la π-calculus, because the scope of identifiers is always global, i.e. each user potentially can access every tweet in the network (in Twitter, for example, it is possible to access the messages sent and received by any user by visiting his/her Twitter page). The possible insertion of the message in the timeline and notification list of the considered account u is regulated by the following *insertion operators*:

– *tweet insertion*: a message m is inserted in the timeline T of an account only if the sender of m is a following of this account

$$T \oplus^F m = \begin{cases} (T, m) & \text{if } m \downarrow_7 \in F \\ T & \text{otherwise} \end{cases}$$

– *notification insertion*: a message m is inserted in the notification list N of an account with username u only if u is mentioned in the text of m, or m is a retweet whose original tweet message has been sent by u, or m is a retweet of a retweet sent by u, or m is a reply of a message sent by u

$$N \oplus^u m = \begin{cases} (N, m) & \text{if } u \in \mathit{mentions}(m \downarrow_4) \\ & \quad \vee\ m \downarrow_5 = u \ \vee\ m \downarrow_6 = u \\ N & \text{otherwise} \end{cases}$$

Rule [N-DELETE] deletes the tweet identified by id and all its retweet from the account that performed the **delete** action (which is the account that emitted such a tweet). The deletion of a message from a list L (which denotes either a timeline or a notification list) is defined by the following operator:

– *tweet deletion*: a message m is deleted from the list L of an account only if id is its identifier or m is a retweet of a message identified by id

$$L \ominus id \ = \ L \backslash \{m \in L \mid m \downarrow_1 = id \ \vee \ m \downarrow_2 = id\}$$

Moreover, retweeting and replying of deleted messages (which may happen when a **delete** is executed after a **search**) are prevented by means of the *block* operator $B\not\downarrow_{id}$, which replaces prefixes $a.B'$ in B by **nil** when a is a **retweet** or a **reply** action having a message m as parameter with $m \downarrow_1$, $m \downarrow_2$ or $m \downarrow_3$ sets to id[6].

The deletion is propagated to the other accounts by rule [N-DELPROPAG].

Retweets are undone by means of rules [N-UNDO] and [N-UNDOPROPAG], that are similar to rules for the **delete** action except for the deletion operator:

– *retweet deletion*: a message m is deleted from the list L of an account only if m has id as identifier of the current message

$$L \oslash id \ = \ L \backslash \{m \in L \mid m \downarrow_1 = id\}$$

[6] The definition of more sophisticated solutions, e.g. based on exception handling, to deal with actions involving deleted messages is left for future investigation. Indeed, the blocking solution used here properly suits the study carried out in this paper.

In this case, it is considered only the first field of the message. Thus, only the retweets identified by id are removed, while other retweets of the same tweet and the tweet itself are not affected by the deletion.

Rule [N-SEARCH-U] allows account u to search for a message satisfying predicate $\mathcal{P}$ in its timeline and notification list. If a message is found, say m, the label produced at network level is $m : \mathbf{found}()$. Rule [N-SEARCH-T] is similar, but it looks for a message in the timeline of another account u', which either is specifically indicated in the target ($t = u'$) or is anyone of the rest of the net ($t = \mathbf{all}$). Once a message is found, rule [N-PAR] permits terminating the search without affecting the other accounts of the network.

Rule [N-FOLLOW1] extends the followings list of account u with username u' when the former is not a follower of the latter; consequently, extends the timeline T with messages (i.e. tweets and retweets) sent by u'. Rule [N-FOLLOW2] is used to let the **follow** action pass without affecting the timeline and the following list of u when this account is already a follower of u'. Rule [N-UNFOLLOW] performs the inverse operations, i.e. it removes u' from F and the messages sent by u' from T. Rule [N-PAR] is used for allowing the whole network to evolve accordingly. Notably, for the sake of simplicity, if the argument u' of actions **follow** and **unfollow** does not correspond to an account of the network, or it is the same account performing such actions, rules [N-FOLLOW] and [N-UNFOLLOW] cannot be applied and, hence, the actions are blocked. In fact, this kind of situations cannot take place in Twitter, where only existing accounts can be object of actions **follow** and **unfollow**.

Finally, rule [N-STR] states that structural congruent nets have the same transitions.