On Stable Labellings and Odd-Length Cycles in Abstract Argumentation Frameworks

Claudia Schulz
Imperial College London

Introduction

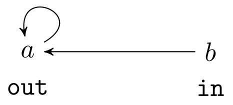
The semantics of abstract argumentation frameworks (AFs) (Dung 1995) can be expressed in terms of argument labellings (Caminada and Gabbay 2009), which assign one of the labels `in` (accepted), `out` (rejected), or `undec` (undecided) to each argument. Most labelling semantics are based on *complete labellings*, which fulfill the conditions that an argument is labelled `in` if and only if it is only attacked by arguments labelled `out` and an argument is labelled `out` if and only if it is attacked by some argument labelled `in`.

The most decisive complete labellings are those which label all arguments as `in` or `out`, and no arguments as `undec`. Such labellings are called *stable labellings* (Caminada and Gabbay 2009). Unfortunately, stable labellings are not guaranteed to exist for all AFs, which is undesirable when using stable labellings as the semantics of choice in an application. Two questions thus arise:

1. Which part of the AF is responsible that no stable labelling exists?
2. What revision to the structure of the AF is necessary to obtain a stable labelling?

Odd-length cycles

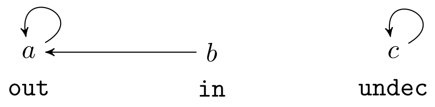
Regarding the first question, it has been shown that AFs which have no stable labellings comprise an odd-length cycle of attacking arguments (Dung 1995). However, it is not the case that every AF comprising an odd-length cycle (of attacking arguments) has no stable labellings, as shown in Figure 1.



$$\begin{tikzcd}
a \arrow[loop above] & b \arrow[l] \\
\texttt{out} & \texttt{in}
\end{tikzcd}$$

Figure 1: An AF comprising an odd-length cycle and its only stable labelling.

Similarly, an AF which has no stable labelling may comprise various odd-length cycles, but not all of them are "responsible" that no stable labelling exists, as demonstrated in Figure 2. The odd-length cycle of argument a attacking itself is not responsible that no stable labelling exists since it is labelled `out` in any complete (and thus in a stable) labelling. In contrast, the only label argument c can have in a complete labelling is `undec`, thus causing the non-existence of a stable labelling for this AF.



$$\begin{tikzcd}
a \arrow[loop above] & b \arrow[l] & c \arrow[loop above] \\
\texttt{out} & \texttt{in} & \texttt{undec}
\end{tikzcd}$$

Figure 2: An AF comprising two odd-length cycles and its only complete/semi-stable labelling.

The first question can thus be further refined to: Which *odd-length cycle* of the AF is responsible that no stable labelling exists?

Responsible odd-length cycles

To answer this question, we make use of the labelling semantics which is closest to stable labellings, namely *semi-stable labellings*, i.e. complete labellings with a minimal (w.r.t. set-inclusion) set of `undec` arguments (Caminada 2006; Baroni, Caminada, and Giacomin 2011). We first note that in every semi-stable labelling of an AF without stable labellings there exists an odd-length cycle whose arguments are all labelled `undec`, for example the odd-length cycle of the self-attacking argument c in Figure 2, which we intuitively identified as the "responsible odd-length cycle.

However, in general there may be various odd-length cycles of `undec` labelled arguments with respect to a semi-stable labelling, but not all of them are "responsible". We say that an odd-length cycle is "responsible" if its structure necessarily has to be changed to obtain a stable labelling. In other words, if the cycle's structure is not changed, the AF will not have a stable labelling no matter which other structural changes are made to the rest of the AF. Consider for example the AF in Figure 3, whose only semi-stable labelling labels all arguments as `undec`. There are thus two odd-length cycles whose arguments are all labelled `undec`, $a - b - c$ and $d - e - f$, but only the first one is a "responsible" odd-length cycle: No matter how the structure of the AF is changed, if the cycle $a - b - c$ is left untouched (not deleting attacks/arguments in it or adding attacks/arguments to it), the resulting AF will not have a stable labelling. In contrast, the structure of the cycle $d - e - f$ does not necessarily have to be changed to yield an AF with a stable labelling: deleting for example the attack from argument b to argument c yields an AF which has a stable labelling.

$$\begin{tikzcd}
\texttt{undec} & & \texttt{undec} & \texttt{undec} & & \texttt{undec} \\
a \arrow[rr, bend left] & & b \arrow[r] \arrow[dl, bend left] & d \arrow[rr, bend left] & & e \arrow[dl, bend left] \\
& c \arrow[ul, bend left] & & & f \arrow[ul, bend left] & \\
& \texttt{undec} & & & \texttt{undec} &
\end{tikzcd}$$

Figure 3: An AF comprising two odd-length cycles and its only semi-stable labelling.

Strongly connected `undec` parts (SCUPs)

The "responsible" odd-length cycles can be formally characterised as odd-length cycles which are part of a *strongly connected* `undec` *part* (SCUP). A SCUP with respect to a semi-stable (or more generally a complete) labelling is a part of an AF such that:

- all arguments are strongly connected, i.e. there is a path (of attacks) from each argument to every other argument in the part,
- all arguments are labelled `undec`, and
- no argument is attacked by an `undec` argument which is not comprised in the part.

The AF in Figure 3 has only one SCUP, namely the odd-length cycle $a - b - c$ which we previously identified as the "responsible" cycle.

An important property of SCUPs is that starting from any `undec` argument in a semi-stable labelling and iteratively following an attack backwards to another `undec` argument will lead to a SCUP. For example, starting from the `undec` argument e in the AF in Figure 3, the "backwards path" $e \leftarrow d \leftarrow b$ leads to a SCUP.

Revising SCUPs

Using the characterisation of responsible parts of an AF as odd-length cycles contained in SCUPs, we can also answer the second question, i.e. what revision to the structure of the AF is necessary to obtain a stable labelling. Note that since an AF encodes some underlying knowledge, it is desirable to use a "minimal" revision which changes only what is necessary to yield a stable labelling.

A naive revision of an AF without stable labellings would be to "break" every odd-length cycle since AFs without odd-length cycles are guaranteed to have a stable labelling (Dung 1995). However, as previously shown it can be sufficient to revise only "responsible" odd-length cycles, i.e. those comprised in a SCUP, to yield an AF that has a stable labelling. Note that revising an odd-length cycle in a SCUP may result in new SCUPs, thus rendering more odd-length cycles "responsible".

In the worst case, iteratively revising odd-length cycles in SCUPs amounts to revising all odd-length cycles in an AF, as shown in Figure 4: Initially a_1 is the only SCUP. Revising the SCUP by breaking the odd-length cycle, e.g. by deleting the self-attack, changes the label of a_1 to `in` in the new semi-stable labelling of the revised AF, and the label of b_1 to `out`. All other arguments remain `undec`. In the revised AF with respect to the new semi-stable labelling, a_2 is now a SCUP. Revising it in the same way as a_1 leads to a_3 being the SCUP, and so on, until reaching a_n which is the last SCUP to be revised. Thus, after revising all n odd-length cycles, the revised AF has a stable labelling which labels all a_i as `in` and all b_j as `out`.

$$\begin{tikzcd}
a_1 \arrow[loop above] \arrow[r] & b_1 \arrow[r] & a_2 \arrow[loop above] \arrow[r] & b_2 \arrow[r] & a_3 \arrow[loop above] & \cdots & b_{n-1} \arrow[r] & a_n \arrow[loop above]
\end{tikzcd}$$

Figure 4: An AF comprising n odd-length cycles, illustrating the worst case for revising SCUPs.

In contrast, in the best case revising odd-length cycles in SCUPs amounts to revising only one odd-length cycle, as demonstrated in Figure 5: The only SCUP is c_1. Revising it by deleting the self-attack of c_1 directly yields a stable labelling where c_1, c_3, and all b_j are labelled `in` and c_2 and all a_i are labelled `out`.

$$\begin{tikzcd}
a_1 \arrow[loop above] \arrow[r] & b_1 \arrow[r] & a_2 \arrow[loop above] \arrow[r] & b_2 \arrow[r] & a_3 \arrow[loop above] & \cdots & b_{n-1} \arrow[r] & a_n \arrow[loop above] \\
c_3 \arrow[u] & c_2 \arrow[l] & c_1 \arrow[loop above] \arrow[l] & & & & &
\end{tikzcd}$$

Figure 5: An AF comprising $n + 1$ odd-length cycles, illustrating the best case of revising SCUPs.

Thus, revising SCUPs minimises the amount of changes necessary to yield a stable labelling as the revision is focused on "responsible" odd-length cycles.

References

Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An Introduction to Argumentation Semantics. *The Knowledge Engineering Review* 26(04):365–410.

Caminada, M., and Gabbay, D. M. 2009. A Logical Account of Formal Argumentation. *Studia Logica* 93(2-3):109–145.

Caminada, M. 2006. Semi-Stable Semantics. In *COMMA'06*, 121–130.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence* 77(2):321–357.