Reduction-based Formal Analysis of BGP Instances

Anduo Wang[1] Carolyn Talcott[2] Alexander J. T. Gurney[1]
Boon Thau Loo[1] Andre Scedrov[1]

University of Pennsylvania SRI International
{anduo,boonloo}@cis.upenn.edu clt@csl.sri.com
agurney@seas.upenn.edu scedrov@math.upenn.edu

Abstract. Today's Internet interdomain routing protocol, the Border Gateway Protocol (BGP), is increasingly complicated and fragile due to policy misconfigurations by individual autonomous systems (ASes). These misconfigurations are often difficult to manually diagnose beyond a small number of nodes due to the state explosion problem. To aid the diagnosis of potential anomalies, researchers have developed various formal models and analysis tools. However, these techniques do not scale well or do not cover the full set of anomalies. Current techniques use oversimplified BGP models that capture either anomalies within or across ASes, but not the interactions between the two. To address these limitations, we propose a novel approach that reduces network size prior to analysis, while preserving crucial BGP correctness properties. Using Maude, we have developed a toolkit that takes as input a network instance consisting of ASes and their policy configurations, and then performs formal analysis on the reduced instance for safety (protocol convergence). Our results show that our reduction-based analysis allows us to analyze significantly larger network instances at low reduction overhead.

1 Introduction

The Internet today runs on a complex routing protocol called the *Border Gateway Protocol* or *BGP* for short. BGP enables Internet Service Providers (ISPs) worldwide to exchange reachability information to destinations over the Internet, and simultaneously, each ISP acts as an autonomous system that imposes its own import and export policies on route advertisements exchanged with its neighbors.

Over the past few years, there has been a growing consensus on the complexity and fragility of BGP routing. Even when the basic routing protocol converges, conflicting policy decisions among different ISPs have led to route oscillation and slow convergence. Several empirical studies (e.g. [12]) have shown that there are prolonged periods in which the Internet cannot reliably route data packets to specific destinations, due to routing errors induced by BGP.

Since protocol oscillations cause serious performance disruptions and router overhead, researchers devote significant attention to BGP stability (or "safety"). A BGP system converges and is said to be safe, if it produces stable routing tables, given any sequence of routing message exchanges. We broadly refer to any route misconfigurations that result in instability as *BGP anomalies* in this paper. To study potential configuration issues with BGP, the network community has studied small network instances.

Sometimes, these come from a single network (the "internal BGP" or "iBGP" case), or they may relate to interaction between different networks ("external BGP" or "eBGP"). These small topology configurations (or "gadgets") serve as examples of safe systems, or counterexamples showing a safety problem such as lack of convergence.

Today, analyzing these gadgets is a manual and tedious process, let alone analyzing actual network instances that are orders of magnitude larger. Researchers check these gadgets by manually constructing "activation sequences" where the nodes make successive routing decisions that form an oscillation. To automate the process, in our prior work [18], we have developed an analysis toolkit using Maude [1] that automates the process of analyzing BGP instances using a *rewriting logic* [13] approach. While automated, this approach can only work for small network instances, since the approach is susceptible to the state explosion problem as the number of nodes increases. To address these challenges, this paper makes the following contributions.

First, we identify the key contributing attributes in BGP routing that lead to eBGP and iBGP anomalies resulting in route oscillations.

Second, we propose an efficient algorithm for reducing BGP instances, so that the network size can be reduced by merging nodes in such a way that the overall convergence properties remain the same. Our reduction uses the well-known *Stable Paths Problem* (SPP) [9] formalism for safety analysis of BGP configurations, where the entire instance is modeled in terms of the router-level topology and each router's policy-induced route preferences. We show how the reduction works for both inter-AS and intra-AS policy configurations using well-known gadgets as examples, and provide formal proofs that the reduction correctly preserves convergence properties in general for any arbitrary BGP instances. The reduction process not only reduces the state size for subsequent analysis, but also provides us the capability to reduce an existing BGP network instance into a known anomaly (i.e. misbehaving gadget), or determine equivalence between two configuration instances.

Finally, using Maude, we develop a tool that (1) takes as input router configurations, (2) extracts the SPP representation of a protocol by generating and comparing all possible routes against each AS policy, (3) applies the reduction step, and (4) performs an exhaustive state exploration on the reduced BGP instance to check for possible configuration anomalies that result in divergence.

Our results show that the reduction-based analysis is much more effective than the prior approach of doing exhaustive search on unreduced instances [18]. The data demonstrate that we have not only gained speed, but also the ability to analyze network instances that were previously infeasible to study.

For example, an instance which would naively take 221193ms to analyze, can now be reduced in 22ms to one which takes only 8ms to analyze, which makes the new method over 7000 times faster (See our technical report [17] for more details). There are also many instances whose analysis was infeasible with the previous approach, but which can now be tackled by reduction. The naive technique was limited to networks of no more than about 20 nodes (at a push, 25) whereas we now have no difficulty in scaling to instances with over a hundred nodes, and with a greater arc density, which are more characteristic of real networks.

2 Analyzing BGP Anomalies

BGP assumes a network model in which routers are grouped into various Autonomous Systems (ASes), each assumed to be under separate administrative control. An individual AS exchanges route advertisements with neighboring ASes using a *path-vector* protocol. Upon receiving a route advertisement, a BGP router may choose to accept or ignore the advertisement based on its *import policy*. If the route is accepted, the node stores the route as a possible candidate. Each node selects among all candidate routes the best route to each destination, based on its local *route preference policy*. Once a best route is selected, the node advertises it to its neighbors. A BGP node may choose to export only selected routes to its neighboring ASes based on its *export policy*. The determination of these three kinds of policy is up to the network operator: BGP allows considerable flexibility. Conflicting policies, within or between ASes, are the cause of protocol oscillation, as the protocol struggles and fails to satisfy all policies at once.

Router-to-router BGP sessions come in two flavors: external BGP (eBGP), which establishes routes between ASes; and internal BGP (iBGP), which distributes routes within an AS. All routers maintain internal state, including their roster of known paths for all destinations, and the list of other routers to whom they are connected. They communicate with one another by exchanging route advertisements. Not all routers communicate directly with routers outside their own AS. If they do, they are *border routers*; if they do not, they are *internal routers*. Additionally, some routers may have a special role as *route reflectors*, collating and distributing route advertisements on behalf of their *clients*, to avoid having to establish pairwise connections between all routers in an AS.

Stage	BGP route selection step
eBGP	**1. Highest LOC_PREF**
	2. Lowest AS path length
	3. Lowest origin type
iBGP	**4. Lowest MED (with same NEXT-HOP AS)**
	5. Closest exit point (lowest IGP cost)
	6. Lowest router ID (break tie)

Table 1. Key attributes in BGP route selection

Every BGP route is endowed with *attributes* that describe it. These are summarized in Table 1. We also characterize these by whether they are primarily associated with eBGP- or iBGP-level routing decisions. Whenever an AS receives a new route, it will compare the attributes of its current available routes (for a given destination) with the new route, and then decide whether the new route is selected as best route. The attributes are listed in the order in which they are compared during route selection: if the routes are tied at any stage, then BGP proceeds to consider the next attribute on the list.

The most important attribute in eBGP route selection is local preference (`LOC_PREF`). This is a value set by each router on routes it receives, according to (arbitrary) rules established by the network operator. If two routes have the same local preference, then the next tiebreaking attribute is the AS path length—the number of ASes through which this route passes—followed by the 'origin' code. The next step is to use the multi-exit discriminator (`MED`) attribute, the most important attribute in iBGP route selection, which

says which individual link is preferred, out of the many links between this AS and its neighbor. If that was not enough to determine a single best route, BGP breaks ties by examining the shortest-path distance to the relevant border router. Finally, if all else fails, it uses the value of each router's unique identifier. This final step is meant to ensure that all possible routes can be placed in a total order, with no two routes being equivalent in preference.

Oscillation anomalies in BGP can be localized to the definition and use of particular attributes. This paper looks at three families of problems.

- In **eBGP anomalies**, routing policy conflicts occur at an inter-AS level. The typical causing attribute is `LOC_PREF`, because it is set arbitrarily at each AS, independently of any other.
- **iBGP anomalies** are limited to a single AS, and associated with `MED`. Due to a quirk in the decision procedure, it is possible for there to be three routes p, q, and r such that p is preferred to q, q to r, and r to p. The router will be unable to settle on a single choice, if there is feedback where its actions cause the visibility of those three routes to change.
- **iBGP-IGP anomalies** result from inconsistency between the semantics of route reflectors, and particular IGP distance values.

We will revisit these anomalies and give formal definitions in Section 4. We will also examine the correctness of network reduction with respect to these anomalies.

3 Network Reduction

Existing network analysis techniques do not scale well: Static analysis [6, 5, 8, 2, 7, 14] by checking combinatorial structure that reflects routing oscillations is normally NP-complete; and dynamic analysis [3, 11] by systematic exploration of the protocol state space will likewise suffer an exponential blow-up as problem size increases. As a result, analysis techniques normally assume an over-simplified BGP model that only covers a portion of the routing anomalies in Section 2.

To address these limitations, we propose network reduction that preservers correctness properties - a process that simplifies network instances. Network reduction can be viewed either as a pre-step prior to formal analysis in order to reduce analysis space; or a model construction step that extracts a simplified model from the real BGP instance.

In network reduction, the basic idea is to incrementally merge two network nodes into one while preserving network properties. To formally define reduction, we need to first represent a BGP instance in an abstract form that also captures each node's routing policy. We choose the extended stable paths problem (SPP) as the formal representation to include both eBGP and iBGP instances.

SPP is a well-established combinatorial model of BGP configurations that captures the outcomes of routing policy—which paths are preferred over which other paths, at each router—while avoiding the need for detailed modeling of the BGP decision process in all its complexity.

We extend SPP to define path preference in a more general way. The extended SPP is then used as the representation to implement reduction. In addition, we provide automatic generation of extended SPP for a BGP instance given its network topology and

high-level routing policy (e.g. how the path attributes are configured/transformed). We will revisit this in Section 5.

3.1 Hierarchical Reduction

The SPP formalism captures the route preferences that exist for all routers, over their routes to a single fixed destination. [1] An SPP instance consists of a graph, together with each router's preferences over paths in the graph. We define this in a more general way than in previous work.

Definition 1 *An extended SPP instance is given by $G = (V, E, d, P, \prec)$, where V is the set of nodes, E is the set of directed arcs, $d \in V$ is the destination node, P is the set of all permitted paths to d, and the binary relation $\prec$ over P indicates when one route is preferred over another. Every path in P must be a simple path (that is, no node appears more than once).*

For a given extended SPP instance G as above, and a node i in V, write P^i for the subset of P consisting of paths from i to d. The SPP definition requires that $\prec$ be a transitive total order on each P^i, but our definition does not enforce that, and supports more routing policies. Routes from different source nodes are incomparable. Conventionally, '$p \prec q$' means that path p is preferred to path q, where both p and q are paths to d from the same source.

In this paper, we will use the symbol '$\circ$' for concatenation of arcs and paths. If (i, j) is an arc in E, and p is a path from j to d, then their concatenation $(i, j) \circ p$ is a path from i to d. Similarly, if p is a path from i to j, and q is a path from k to l, and (j, k) is an arc in E, then the concatenation is $p \circ (j, k) \circ q$ or just $p \circ q$.

This combinatorial definition washes away the some important features about how BGP operates and how paths are chosen: in particular, the distinction between external and internal BGP. The eBGP/iBGP distinction is critical for our reduction technique, because it is based on the observation that certain kinds of anomaly can be 'localized' to one or the other mode. Our reductions will operate on the iBGP level first, for each AS. After iBGP simplification, we simplify eBGP by reducing the extended SPP instance for the remaining network. This ordering also allows certain kinds of inconsistency, that can only occur in iBGP, to be detected and handled; we do not need to contaminate our other reductions with knowledge of these special cases. Since our reduction method includes steps that are specific to one or the other mode of operation— we assume that, in reduction, all we are faced with are extended SPP instances, derived from BGP configurations.

3.2 Network Reduction

This subsection proposes sufficient conditions for two BGP nodes to be 'unifiable', meaning that they can be merged into one node. The reduction proceeds by repeatedly (1) locating two unifiable nodes, and rewriting their local configuration, and (2) rewriting the remainder of the BGP instance to reflect that local change. In the following, assume we are working with a given extended SPP instance $G = (V, E, d, P, \prec)$.

[1] Route preferences are configurable separately for each destination, so this assumption focuses the analysis rather than limits it.

Locate unifiable nodes

We identify two special cases of unifiable nodes, which we call *duplicate* and *supplementary*. We first define an auxiliary notion: "node rewrite", based on which unifiable node conditions are defined.

Definition 2 *For two nodes i, k in V,* rewrite *i to k by rewriting P^i and $\prec$ as follows:*

1. *Check for any path p in P on which i and k both occur, if they occur only in an adjacent position, then proceed to the next step, otherwise abort the rewrite.*
2. *For every path p in P^i, if i or ik occurs, replace it by k.*
3. *For every two distinct paths p and q in P^i, that rewrite to p' and q' respectively, check whether p' and q' are equal. If they are, then abort the rewrite; otherwise, proceed to the next step.*
4. *Every preference $p \prec q$, where p and q are in P^i and rewrite to p' and q' respectively, becomes $p' \prec q'$.*

Step 1 ensures that if there is *any* permitted path on which i and k both occur, with some intervening nodes between them, then they are not considered for rewriting (unification). Therefore, after the first step of rewriting, the paths in P^i will still be simple (k will not occur twice). Step 3 ensures that after rewriting, no two paths in P^i can collapse into one. Based on this rewriting notion, the two unifiable node conditions are as follows.

Definition 3 *Two nodes i, j in V are* unifiable *if i is supplementary for j, or i and j are duplicate, where:*

1. A node i is *supplementary* for j if:
   1. i can be rewritten to j as defined in Definition 2.
   2. For every path p in P^i, there is some path q in P^j such that p and q are equal after rewriting.
   3. Whenever $p_1 \prec p_2$ in P^i, there are paths q_1 and q_2 in P^j such that $q_1 \prec q_2$; p_1 and q_1 are equal after rewriting; and p_2 and q_2 are equal after rewriting.
4. Two nodes i and j in V are *duplicate* if each is supplementary for the other.

Reduce BGP Instance

After locating two unifiable nodes i and j, we rewrite the entire extended SPP to reflect this unification. This completes one network reduction step.

First define a function θ_{ij} from V to $V \setminus \{i\}$ by $\theta_{ij}(i) = \theta_{ij}(j) = j$, and $\theta_{ij}(x) = x$ for all x not equal to either i or j. This function induces corresponding maps on E and P, as follows.

Definition 4 *If i and j are unifiable nodes in V, then G may be* reduced *to $G' = (V', E', d, P', \prec')$, where*

- $V' = V \setminus \{i\}$
- $E' = \{(\theta_{ij}(u), \theta_{ij}(v)) \mid (u, v) \in E \setminus \{(i, j), (j, i)\}\}$
- *P' consists of all paths in P after rewriting each node according to θ_{ij}, and eliding any (j, j) arc.*
- *$p' \prec' q'$ if and only if $p' \neq q'$ and there exist paths p and q in P such that p rewrites to p', q rewrites to q', and $p \prec q$.*

3.3 Examples: Reducing eBGP and iBGP Instances

We now illustrate the intuition of network reduction by applying reduction to various eBGP and iBGP instances.

Example 1 *Reducing eBGP instances* Two eBGP instances called *Bad gadget* and *Good gadget* are shown on the left of Figure 1. The topology of each eBGP instance is given by the network graph, whereas the routing policies are shown by the path preferences indicated alongside each network node. In each list, the more preferred paths are at the top, and paths that do not appear are not permitted. For example, in the good gadget, the policy for node `1` says it has two permitted paths, `1 3 0` and `1 0`, where `1 3 0` is preferred to `1 0`.

In both gadgets, nodes `3`,`4` are unifiable nodes according to Definition 3. After reduction, these nodes are merged into one, shown on the right hand side of Figure 1.

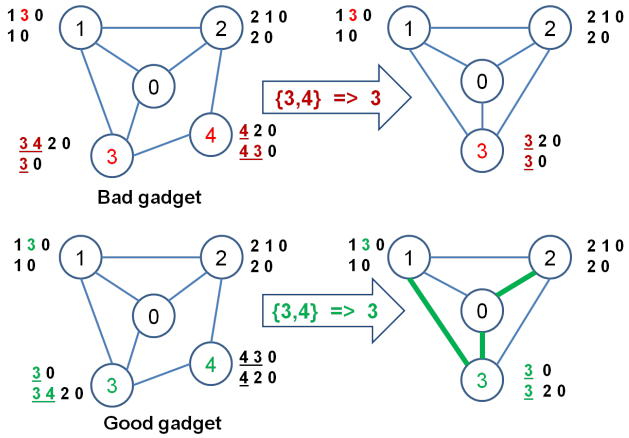

Fig. 1. Reducing bad/good gadget makes it easier to detect divergence/prove safety.

The reason why the bad gadget is called 'bad' is that it suffers from permanent route oscillation: the preferences are incompatible, there is no stable solution, and the iterative attempt to find one does not terminate. The 'dispute wheel' pattern alluded to above is what causes the badness, and after reduction this pattern becomes clearer. In the reduced bad gadget instance, we can see that each of the three outer nodes prefers an indirect path (around the cycle and then in) over a direct one (straight to the destination). This is an order-three dispute wheel. The pattern was present in the original instance, but obscured by the presence of node `4`. On the other hand, the 'good' gadget has a unique stable solution, which is found by iteration. We can identify the solution on the reduced instance (shown here in green), and the original instance also converges.

In addition to good and bad gadget, our technical report [17] shows an eBGP instance that is not reducible, and the reduction of an iBGP instance.

4 Correctness of Network Reduction

We have identified three types of routing anomalies in Section 2, and associated each of them with particular BGP attributes. In this section, we examine sufficient conditions by which each of these three can be avoided. These are *safety*, the standard property for convergence of a path-vector routing system; *acyclic preference*, for ensuring that iBGP configurations express a consistent choice function; and *IGP-iBGP consistency*, for avoiding intra-AS oscillation. We then show that our reduction is sound with respect to preservation of the first two properties, but it does not always preserve the third. Therefore, the third condition needs to be checked separately.

4.1 eBGP Correctness

The eBGP correctness property we consider is safety [15, 10]. The progress of the BGP algorithm towards a solution depends on the timing of messages and other non-deterministic factors: we want to ensure that every execution schedule will result in a routing solution being found, regardless of the asynchronous nature of the protocol. The final state is characterized by *stability*, meaning that no future messages will affect which best paths are selected by each router.

Definition 1. *A BGP instance is **safe**, if under all possible executions, it converges to a stable state, where the best routes selected by all the routers form a policy-compliant routing tree.*

We show that our reduction preserves safety, using a structure called the *path digraph* [15]. This is derived from an SPP instance $(V, E, d, P, \prec)$. Compared with the extended SPP which is used to define reduction, SPP requires an additional constraint: $\prec$ totally orders each P^i where i is a node in V. This holds for instances which are restricted to the 'eBGP' attributes, plus the router identifier, in Table 1.

Definition 5 *Let $G = (V, E, d, P, \prec)$ be an SPP instance. The path digraph is a graph whose nodes are the elements of P, and where there is an arc (p, q) from p to q if either of these two cases holds:*

1. *If $q = r \circ p$ for some path r, there is a 'transmission arc'.*
2. *If p and q are two paths in P^i and $p \prec q$, there is a 'preference arc'.*

If the digraph is acyclic then the SPP has a unique stable solution, which can be found by iteration from any starting state. We will call an SPP instance *cyclic* (or *acyclic*) if its path digraph is cyclic (or acyclic).

The following proposition 1, proved by Sobrinho [15], relates cyclicity of the digraph to safety of the SPP, and therefore of the BGP configuration it represents.

Proposition 1. *If a SPP instance is acyclic, then it is safe. If an SPP instance is cyclic, then we can construct an execution trace that exhibits route oscillation.*

Our main result (Lemma 1) is that our reduction technique transforms cyclic SPPs into cyclic SPPs, and acyclic SPPs into acyclic SPPs. This means that we never have false positives or false negatives, with respect to this safety property, after applying the reduction.

Lemma 1. *Let $G = (V, E, d, P, \prec)$ be an SPP instance, containing unifiable nodes u and v, and let $G' = (V', E', d, P', \prec')$ be the result of applying the procedure of Definition 4 to unify those two nodes. Then G is cyclic if and only if G' is cyclic.*

Proof. See technical report [17] for more details. □

Finally, the following theorem proves that network reduction is sound: to analyze G for safety, it is sufficient to analyze its reduction G'.

Theorem 1. *If G' is acyclic then G is safe; If G' is cyclic then in running G, there exists at least one execution trace that exhibits route oscillation.*

Proof. Obvious from Lemma 1 and Proposition 1. □

4.2 iBGP correctness: Cyclic iBGP Route Preference

As previously noted, use of the `MED` attribute means that routes might not be totally ordered, and therefore Proposition 1 is inapplicable. We handle this case by employing a more general notion of route selection in our analysis, and can show that our reduction *does* preserve these kinds of preference cycle. The details are in technical report [17].

4.3 iBGP Correctness: IGP-iBGP Consistency Property

While BGP can choose the correct egress point in an AS, for each destination, establishment of the intra-AS path to that border router is the responsibility of another protocol (an interior gateway protocol or IGP). Problems can occur if the iBGP configuration does not match the distance values used in the IGP. Our network reduction is designed for analysis BGP routing policies, and is unaware of IGP-iBGP inconsistency (see technical report [17]). Therefore, to ensure the soundness of analysis, one should check IGP-iBGP consistency before applying network reduction, using pre-existing methods from the literature [16, 4].

5 Network Reduction in Maude

To validate our reduction method, we have extended our library for analysis of BGP configurations [18] to support automatic abstraction from dynamic (BGP) configurations to static (extended SPP) configurations, reduction based on SPP configurations, and integration with dynamic exhaustive search analysis. Using the original library BGP instances up to 25 nodes have been successfully analyzed in *minutes*. Using our reduction technique, we are able to reduce and analyze various 100 nodes BGP instances within *seconds*. Our extended library consists of the following three components:

- **Dynamic network representation** For a BGP instance, we require users to input routing policies, i.e., the values of the BGP attributes that cause anomalies. We also require users to input the network topology. Based on the routing policy and topology, we automatically generate the dynamic representation of the BGP instance. The dynamic representation includes configurations (snapshots of an executing instance) and rewrite rules describing a router's actions during execution of the BGP protocol. The dynamic representation can be used to compute the complete set of permitted paths, and route selection information.

- **Static network representation** While the dynamic representation is good for simulating the dynamic behavior of a BGP system, it is not the right representation for network reduction. Thus we introduce a static representation of BGP instances corresponding to the extended SPP instance (Definition 1). For each router, its static representation consists of its complete set of permitted paths, and route selection result given any sub-set of the permitted paths. Our library provides functions to compute the static representation from the dynamic initial network state.
- **Network reduction on static representation** Our library implements the network reduction process described in Definition 4 that applies to the static (extended SPP) representation.

Our library is implemented in Maude [1], a language and tool based on rewriting logic. Rewriting logic [13] is a logical formalism that is based on two simple ideas: states of a system can be represented as elements of an algebraic data type, and the behavior of a system can be given by transitions between states described by local rewrite rules. A rewrite rule has the form '$t \implies t'$ if c' where t and t' are patterns (terms possibly containing variables) and c is a condition (a boolean term). Such a rule applies to a system state s if t can be matched to a part of s by supplying the right values for the variables, and if the condition c holds when supplied with those values. In this case the rule can be applied by replacing the part of s matching t by t' using the matching values for variables in t'. Maude provides a high performance rewriting engine featuring matching modulo associativity, commutativity, and identity axioms. Given a specification S of a concurrent system, Maude can execute this specification, allowing one to observe some possible behaviors of the system. One can also use the search functionality of Maude to check if a state meeting a given condition can be reached during any system execution.

The dynamic representation is a small extension of [18] to account for the MED attribute. In this paper we only discuss generation of the static representation and the implementation of the reduction process.

5.1 Computing the Static BGP Representation

We recall that the dynamic representation of a BGP router has the form `[rid : asid |Nb: nbrs,LR: routes ,BR: best]` where `rid : asid` is called the `NodeInfo` with `rid` the router ID, and `asid` the AS ID. The remaining three arguments represent the routers state: `nbrs` is a list of neighbor router IDs, `routes` is a list of routes, and `best` is the best route.

Recall that in Definition 4, we apply the network reduction to the static representation of a BGP system $G = (V, E, d, P, \prec)$. In this representation we need the following information: (1) the complete set of permitted paths P that the routers could ever generate in protocol execution; and (2) the $\prec$ relation that determines how each router selects the best route, given an arbitrary subset of permitted paths. To capture P and $\prec$, we introduce the static representation of a BGP system using the Maude constructor declaration:

```
op [_|Nb:_,perPath:_,pref:_] : NodeInfo List{NodeInfo} List{route} List{sel-fun}
    -> absNode .
```

Similar to the dynamic representation, the first two arguments (indicated by underscores) specify the router's ID, AS and neighbor information. What is different is the second two attributes: rather than keeping the dynamic routing table and best route attributes, we have the static permitted paths attribute `perPath:`, and the route preference attribute `pref:`. The value of `perPath:` is the list of paths that can be computed during BGP execution, and the value of `pref:` represents the preference function as a list of pairs, each consisting of a route set and the selected route.

A BGP system's static representation is computed from the specification of the dynamic representation in two steps. First, the complete set of permitted paths is computed by simulating route exchanges and computation on the dynamic representation using the the rewrite rule `compute-spp`:

```
rl [compute-spp]:
[from (S1 : AS1) to S2 : (S3 : AS3),lf2,[asp1],med1,S4]
[S2 : AS2 |Nb: nodes2, LR: lr2, BR: nilRoute ]
=>
if ((occurs(import((S1 : AS1),(S2 : AS2),((S3 : AS3),lf2,[asp1],med1,S4)),lr2)) or
    import(...) == nilRoute)
 then [S2 : AS2 |Nb: nodes2, LR: lr2, BR: nilRoute ]
 else
  [S2:AS2|Nb: nodes2,
          LR: update(import(...),lr2),
          BR: nilRoute ]
  generateMsg((S2:AS2),nodes2,export(import(...)))
fi .
```

Here, the left-hand matches a router `S2` and a route message sent from its neighbor `S1`. The right-hand side says that `S2` computes a new route `import(...)`, and if either of the two conditions `occurs(import(...), lr2)` or `import(...)==nilRoute` holds, that is, if either the new route `import(...)` is already in routing table `lr2`, or if the new route is filtered out according to `S2`'s routing policy, `S2` is unchanged, and the routing message on the left-hand is consumed. Otherwise, the new route is inserted into the routing table (`update(import(...),lr2)`), and `S2` applies its export policy `export(import(...))` and then (if allowed by export policy and export does not result in `nilRoute`) `S2` re-advertises this new route to all of its neighbors `nodes2`. Compared with the normal BGP protocol execution, this rule is simpler in the sense that it does not perform best route selection: Note that `BR:` is kept blank. Normal BGP execution is non-deterministic—depending on the result of route selection, one of three different types of actions are taken [18], and the system may converge to different final states or not terminate (route oscillation may happen due to conflicting best route selection). However, the process defined by rule `compute-spp` always terminates with the same final state, when the complete sets of permitted paths of all nodes are generated.

Second, based on the permitted paths, the route selection function `pref:` is computed as follows:

```
eq compSPP ([S1 : AS1 |Nb: nodes1,LR: lr1, BR: nilRoute] Network) =
[ S1 : AS1 |Nb: nodes1, perPath: lr1, pref: compSPPNode(lr1)] compSPP(Network) .
```

`compSPP` converts each dynamic router representation `[S1:AS1|Nb:_,LR:_, BR:_]` in the network to its static form `[ S1:AS1 |Nb:_, perPath:_, pref:_]`. The critical part is to compute route selection `compSPPNode(lr1)`, given the complete set of subsets of the permitted paths `lr1`, by applying the best route selection function `select` to each subset. The function `select` is defined in terms of path attributes.

As example, we show here the encoding of the two eBGP attributes LOCAL_PREF and AS_PATH as follows:

```
op select : List{route} -> List{route} .
eq select(lr1) =
    select-as(select-lf(lr1, best-lf(lr1)),
              best-as(select-lf(lr1, best-lf(lr1)))) .
```

Here `select` first invokes `best-lf` to compute the lowest (best) LOCAL_PREF value in the permitted paths `lr1`, then `select-lf` selects from `lr1` the set of routes with this lowest LOCAL_PREF value. Next, from these remaining routes, `select` invokes `best-as` to compute the best AS value and `select-as` to select the set of routes with such best AS value.

5.2 Reduction by Merging All Pairs of Unifiable Nodes

To reduce a BGP instance, we take its static representation - a set of routers of the form `[ S1:AS1 |Nb:_, perPath:_, pref:_]` as input, and repeatedly merge pairs of unifiable nodes. For each router `S1` in the Network, we look for its unifiable nodes, if such nodes exist, we unify `S1` with the first unifiable node `S2`, and transform the rest of the network according to Definition 4 (e.g. the neighbors of `S1,S2` now become neighbors of `S1`). This reduction process is implemented by the function `mergeDupEach`.[2]

First, `mergeDupEach` implements the process of unifying node `S1` and its first-found unifiable node as follows:

```
eq mergeDupEach([(S1:AS1) |Nb:nodes1, perPath:lr1, pref:lsel-fun1],
                ([(S2:AS2)|Nb:nodes2, perPath:lr2, pref:lsel-fun2] C))
  =
  if (size(([(S1 : AS1) |Nb: nodes1, perPath: lr1, pref: lsel-fun1] unify
            [(S2 : AS2) |Nb: nodes2, perPath: lr2, pref: lsel-fun2])) == 1)
    then
      (([(S1:... ] unify [(S2:...]) C)
    else
      ([(S2:...] mergeDupEach([(S1:...], C))
  fi .
```

Here, the `if` condition tests if `S1,S2` are unifiable, and `mergeDupEach` tests nodes in the network `C` until a unifiable node is found. Then `mergeDupEachEachRW` is invoked.

```
eq mergeDupEachRW(abn, C) =
   replaceNode(mergeDupEach(abn,C), get-NodeInfo(abn), findNodeInfo(abn, C))
```

Here, network `C` is transformed by replacing information relating to `abn` by that of `abn`'s first unifiable node `findNodeInfo(abn, C)`. The specific transformation is as follows:

```
eq replaceNode ([S0:AS)|...] C, (S1:AS1) , (S2:AS2)) =
    [(S0 : AS) |Nb: removeRepeatedNB (...,(S1:AS1),(S2:AS2)),
                perPath: (replacePerPath(...,(S1:AS1),(S2:AS2))),
                pref: replacePref(...,(S1:AS1), (S2:AS2))]
    replaceNode (C, (S1:AS1), (S2:AS2)) .
```

[2] Obviously, reduction always terminates. However, how the order of merging nodes affects the reduction process—whether reduction always converges to the same reduced network—is the subject of ongoing work, but does not affect correctness.

Here, each node `S0` in the network is transformed by rewriting its neighboring table (`NB:`), permitted path (`perPath:`), and route selection function (`pref:`).

Finally, putting it all together, `mergeDup` specifies the reduction process on the entire network `C` as follows:

```
eq mergeDup(S1 Oidl, C) =
  mergeDupEachRW(get-Node(S1, mergeDup(Oidl, C)),
                 mergeDup(Oidl, C) - get-Node(S1, mergeDup(Oidl, C))) .
```

Here `mergeDup` takes two inputs. The first argument `S Oidl` is the list of router IDs, and the second argument is the list of routers `[ S1:AS1 |Nb:_, perPath:_, pref:_]`. `mergeDup(Oidl, C)` denotes the set of remaining routers after reducing all nodes other than `S1`; if `S1` is in these remaining routers, then `get-Node(S1, mergeDup(Oidl, C))` denotes `S1` itself, otherwise (that is, if `S1` is removed in the reduction) the value is set to nil. In either case, `mergeDup(Oidl, C) - get-Node(S1, mergeDup(Oidl, C))` denotes the remaining routers other than `S1` after reducing all routers except `S1`. Based on these notions, the recursive definition of `mergeDup` says that, to merge all unifiable nodes, we only need to merge node `S1` into the routers that (1) are already reduced among themselves; and (2) do not contain `S1` itself.

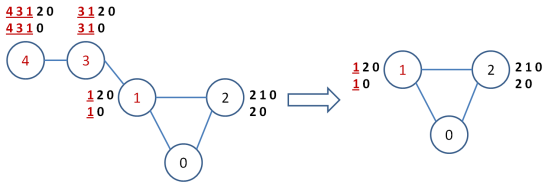

Fig. 2. Reduction example

As an example, to perform network reduction to the network on the left of Figure 2, we execute in Maude as follows:

```
red mergeDup(N1 N2 N3 N4, Network)
```

Where `Nodes = N1 N2 N3 N4` and `Network = [N0:0 |Nb:_,LR:_,BR:_] [N1 : 1 |...] ... [N5 : 5 |...]`. The result is as follows:

```
[N0 : 0 |Nb: (N1 : 1) (N2 : 2),perPath: nil,pref: nil]
[N1 : 1 |Nb: N2 : 2,
         perPath:(N0 : 0,200,[2 0],[1,2],N2)
                 (N0 : 0,100,[0],[1,0],N0),
         pref: ...]
[N2 : 2 |Nb: N1 : 1, ...]
```

As expected, nodes `N3,N4` are merged into `N2`.

6 Evaluation

In this section, we provide an empirical study to quantify the benefits of network reduction, by comparing the computation time required in safety analysis with and without network reduction.

Our safety analysis was performed via an exhaustive search strategy using Maude, as described in [18]. Oscillation is detected if the same best route is selected multiple times during protocol execution. To detect such recurring best routes, we use Maude

to run the actual path vector protocol used in BGP, and simulate all possible sequences in which ASes receive routes. At each node, we use a monitor object to track the best routes that have been previously selected. We also attempt to apply our reduction technique and perform such analysis on the reduced version.

For the BGP instance shown in Figure 2, we note that in the reduced network (right), our analysis tool detects the same route oscillation pattern found in the original network (left), while requiring significantly less state space (reduction from `956` to `35`) and analysis time (`320ms` to `8ms`). In addition, we evaluate three common scenarios [10]: `Bad` gadget that exhibits permanent oscillation, `Disagree` transient oscillation, and `Good` gadget that is safe and no oscillation. The data table in this section shows the analysis results for `Bad` gadget scenario, indicating the performance requirements for the ordinary exhaustive search and for the reduction alternative, as well as the final safety outcome. For more details on the other scenarios, please see our technical report [17].

	Bad (reduced)	**Bad-10**	**Bad-20**	**Bad-53**	**Bad-83**	**Bad-102**
Search (Time)	30510ms	Unknown	Unknown	Unknown	Unknown	Unknown
SPP generation (Time)	0ms	3ms	44ms	134ms	246ms	273ms
Reduction (Time)	0ms	8ms	49ms	146ms	541ms	595ms
Search(State)	11118	Unknown	Unknown	Unknown	Unknown	Unknown
Oscillation?	Yes	Yes	Yes	Yes	Yes	Yes

Table 2. Network Instances that Reduces to Bad Gadget

Our analysis was carried out on a Intel 2.40GHz dual-core machine with 1.9GB memory, running Maude v2.4 on the Debian 5.0.6. operating system. Table 2 shows the analysis results for eBGP instances where the network size ranges from 10 (`Bad-10`) to 102 (`Bad-102`). For each network size, we embed in a bad gadget. After applying the reduction process, all BGP instances are reduced to a single bad gadget `Bad (reduced)`. For each entry, `Unknown` means that the analysis cannot be completed within reasonable time (after running Maude for several hours).

We make the following observations from our results. First, reduction requires minimal time. Even for a large network of 102 nodes, reduction can be completed within one second. As input to the reduction process, the SPP formalism for a BGP instance is extracted as described in Section 5 where a static representation (corresponding to the SPP) is computed by simulating on the instance's dynamic representation (corresponding to the snapshot state). This is also an efficient process, requiring less than 300ms for the largest network. Overall, network reduction results in significant savings in both state and execution time during safety analysis. For example, while it was previously infeasible to complete the analysis of any network beyond 10 nodes due to the state explosion problem (depicted by `Unknown`), the reduced BGP instance can be analyzed in around 300 seconds (and 11118 states).

In our technique report [17], we present a similar comparison of analysis overhead for network instances that have the disagree and good gadget embedded. We similarly observe significant state and execution time savings via the use of reduction.

7 Conclusion

In this paper, we present a technique to reduce BGP instances, such that safety analysis can be performed efficiently on large networks. We prove correct our reduction

technique, develop a reduction and BGP analysis tool using Maude, and demonstrate its effectiveness at reducing the state space and execution time required for analyzing BGP instances. As future work, we are (1) exploring the use of our tool on larger case studies drawn from real network configurations, (2) making the tool available with documentation, (3) optimizing the formal representation for more efficient analysis, and (4) possibly extending the library to detect iBGP cyclic preference, and IGP-iBGP inconsistency.

Acknowledgment This research is funded in part by NSF grants (CCF-0820208, CNS-0830949, CNS-0845552, CNS-1040672, TC-0905607 and CPS-0932397), AFOSR MURI grant No. FA9550-08-1-0352, and ONR grant N00014-11-1-0555.

References

1. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.
2. N. Feamster, R. Johari, and H. Balakrishnan. Implications of autonomy for the expressiveness of policy routing. In *ACM SIGCOMM*, 2005.
3. A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *ACM SIGCOMM*, 2004.
4. A. Flavel, M. Roughan, N. Bean, and A. Shaikh. Where's Waldo? Practical Searches for Stability in iBGP. In *Proc. International Conference on Network Protocols (ICNP)*, October 2008.
5. L. Gao, T. G. Griffin, and J. Rexford. Inherently safe backup routing with BGP. In *IEEE INFOCOM*, 2001.
6. L. Gao and J. Rexford. Stable Internet routing without global coordination. In *ACM SIGMETRICS*, 2000.
7. T. G. Griffin. The stratified shortest-paths problem. In *COMSNETS*, 2010.
8. T. G. Griffin, A. Jaggard, and V. Ramachandran. Design principles of policy languages for path vector protocols. In *ACM SIGCOMM*, 2003.
9. T. G. Griffin, F. B. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE Trans. on Networking*, 10:232–243, 2002.
10. T. G. Griffin and G. Wilfong. An analysis of BGP convergence properties. In *SIGCOMM*, 1999.
11. A. Haeberlen, I. Avramopoulos, J. Rexford, and P. Druschel. NetReview: Detecting when interdomain routing goes wrong. In *NSDI*, 2009.
12. C. Labovitz, G. Malan, and F. Jahanian. Internet Routing Instability. *TON*, 1998.
13. J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
14. M. Schapira, Y. Zhu, and J. Rexford. Putting BGP on the right path: A case for next-hop routing. In *ACM SIGCOMM HotNets*, Oct. 2010.
15. J. Sobrinho. Network routing with path vector protocols: theory and applications. In *SIGCOMM*, 2003.
16. M. Vutukuru, P. Valiant, S. Kopparty, and H. Balakrishnan. How to Construct a Correct and Scalable iBGP Configuration. In *IEEE INFOCOM*, Barcelona, Spain, April 2006.
17. A. Wang, C. Talcott, A. J. T. Gurney, B. T. Loo, and A. Scedrov. Reduction-based formal analysis of BGP instances. University of Pennsylvania Department of Computer and Information Science Technical Report `http://netdb.cis.upenn.edu/papers/bgpMaudeTR.pdf`.
18. A. Wang, C. Talcott, L. Jia, B. T. Loo, and A. Scedrov. Analyzing bgp instances in maude. In *FMOODS-FORTE*, 2011.