Batch NFS

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Tanja Lange
Technische Universiteit Eindhoven

In this talk $\log L$ means $(1 + o(1))(\log N)^{1/3}(\log \log N)^{2/3}$.
L is often written "$L_N(1/3)$" or "$L_N(1/3)^{1+o(1)}$".

Exponents of L in this talk are limited to $10^{-6}\mathbf{Z}$.

Rigorously proven? Ha ha ha.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

Batch NFS

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Tanja Lange
Technische Universiteit Eindhoven

In this talk $\log L$ means $(1+o(1))(\log N)^{1/3}(\log\log N)^{2/3}$.
L is often written "$L_N(1/3)$" or "$L_N(1/3)^{1+o(1)}$".

Exponents of L in this talk are limited to $10^{-6}\mathbf{Z}$.

Rigorously proven? Ha ha ha.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Batch NFS

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Tanja Lange
Technische Universiteit Eindhoven

In this talk $\log L$ means $(1+o(1))(\log N)^{1/3}(\log\log N)^{2/3}$.
L is often written "$L_N(1/3)$" or "$L_N(1/3)^{1+o(1)}$".

Exponents of L in this talk are limited to $10^{-6}\mathbf{Z}$.

Rigorously proven? Ha ha ha.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

FS

ernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

ange
che Universiteit Eindhoven

---

alk log L means
$))(\log N)^{1/3}(\log\log N)^{2/3}$.
en written
3)" or "$L_N(1/3)^{1+o(1)}$".

nts of L in this talk
ed to $10^{-6}\mathbf{Z}$.

sly proven? Ha ha ha.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example
dnssec-
is signed

is at Chicago &
siteit Eindhoven

siteit Eindhoven

neans
$/3(\log\log N)^{2/3}$.

$_N(1/3)^{1+o(1)}$".

this talk
$^6\mathbf{Z}$.

? Ha ha ha.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP

`dnssec-deployme`

is signed by an RS

ago &
hoven

hoven

$N)^{2/3}$.

$1)$".

a.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 ke

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org`
is signed by an RSA-1024 key
signed by an RSA-2048 key
signed by `org`'s RSA-1024 key
signed by an RSA-2048 key
signed by a root RSA-1024 key

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key signed by a root RSA-1024 key signed by an RSA-2048 key.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key signed by a root RSA-1024 key signed by an RSA-2048 key.

Most "DNSSEC" signatures follow a similar pattern.

2003 Shamir–Tromer, 2003 Lenstra–Tromer–Shamir–Kortsmit–Dodson–Hughes–Leyland, 2005 Geiselmann–Shamir–Steinwandt–Tromer, 2005 Franke–Kleinjung–Paar–Pelzl–Priplata–Stahlke, etc.: RSA-1024 is breakable in a year by an attack machine costing $<10^9$ dollars.

So the Internet switched to RSA-2048, and we no longer care about RSA-1024 security, right?

Wrong!

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key signed by a root RSA-1024 key signed by an RSA-2048 key.

Most "DNSSEC" signatures follow a similar pattern.

Another example: SSL has used many millions of RSA-1024 keys. Imagine that an attacker has recorded tons of SSL traffic.

amir–Tromer, 2003
-Tromer–Shamir–
t–Dodson–Hughes–
2005 Geiselmann–
Steinwandt–Tromer, 2005
Kleinjung–Paar–Pelzl–
–Stahlke, etc.: RSA-1024
able in a year by an attack
costing $<10^9$ dollars.

nternet switched to
48, and we no longer care
SA-1024 security, right?

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key signed by a root RSA-1024 key signed by an RSA-2048 key.

Most "DNSSEC" signatures follow a similar pattern.

Another example: SSL has used many millions of RSA-1024 keys. Imagine that an attacker has recorded tons of SSL traffic.

Users se
1. "The
more tha
2. "The
off-the-s
attacker
3. For s
switch k
the atta

mer, 2003
hamir–
-Hughes–
selmann–
lt–Tromer, 2005
-Paar–Pelzl–
etc.: RSA-1024
ear by an attack
$<10^9$ dollars.

vitched to
e no longer care
security, right?

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key signed by a root RSA-1024 key signed by an RSA-2048 key.

Most “DNSSEC” signatures follow a similar pattern.

Another example: SSL has used many millions of RSA-1024 keys. Imagine that an attacker has recorded tons of SSL traffic.

Users seem uncon
1. “The attack m
more than this RS
2. “The attack m
off-the-shelf; it's o
attackers building
3. For signatures:
switch keys every
the attack machin

, 2005
zl–
-1024
attack
rs.

r care
ght?

Example: The IP address of
`dnssec-deployment.org`
is signed by an RSA-1024 key
signed by an RSA-2048 key
signed by `org`'s RSA-1024 key
signed by an RSA-2048 key
signed by a root RSA-1024 key
signed by an RSA-2048 key.

Most "DNSSEC" signatures
follow a similar pattern.

Another example: SSL has used
many millions of RSA-1024 keys.
Imagine that an attacker has
recorded tons of SSL traffic.

Users seem unconcerned:

1. "The attack machine cos
more than this RSA key is w

2. "The attack machine isn'
off-the-shelf; it's only for
attackers building ASICs."

3. For signatures: "We
switch keys every month, an
the attack machine takes a y

Example: The IP address of
`dnssec-deployment.org`
is signed by an RSA-1024 key
signed by an RSA-2048 key
signed by `org`'s RSA-1024 key
signed by an RSA-2048 key
signed by a root RSA-1024 key
signed by an RSA-2048 key.

Most "DNSSEC" signatures follow a similar pattern.

Another example: SSL has used many millions of RSA-1024 keys. Imagine that an attacker has recorded tons of SSL traffic.

Users seem unconcerned:

1. "The attack machine costs more than this RSA key is worth."

2. "The attack machine isn't off-the-shelf; it's only for attackers building ASICs."

3. For signatures: "We switch keys every month, and the attack machine takes a year."

Example: The IP address of `dnssec-deployment.org` is signed by an RSA-1024 key signed by an RSA-2048 key signed by `org`'s RSA-1024 key signed by an RSA-2048 key signed by a root RSA-1024 key signed by an RSA-2048 key.

Most "DNSSEC" signatures follow a similar pattern.

Another example: SSL has used many millions of RSA-1024 keys. Imagine that an attacker has recorded tons of SSL traffic.

Users seem unconcerned:

1. "The attack machine costs more than this RSA key is worth."

2. "The attack machine isn't off-the-shelf; it's only for attackers building ASICs."

3. For signatures: "We switch keys every month, and the attack machine takes a year."

Real quote: "DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key."

e: The IP address of
-deployment.org
l by an RSA-1024 key
y an RSA-2048 key
y org's RSA-1024 key
y an RSA-2048 key
y a root RSA-1024 key
y an RSA-2048 key.

DNSSEC" signatures
similar pattern.

example: SSL has used
illions of RSA-1024 keys.
that an attacker has
l tons of SSL traffic.

Users seem unconcerned:

1. "The attack machine costs more than this RSA key is worth."

2. "The attack machine isn't off-the-shelf; it's only for attackers building ASICs."

3. For signatures: "We switch keys every month, and the attack machine takes a year."

Real quote: "DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key."

Continua
despite h
broken a
fact, the
estimate
of a 700
breaking
would ne
amounts
power in
be detec
single ke
estimate
safely us
least the

address of
ent.org
SA-1024 key
-2048 key
SA-1024 key
-2048 key
RSA-1024 key
-2048 key.

signatures
ttern.

SSL has used
RSA-1024 keys.
ttacker has
SSL traffic.

Users seem unconcerned:

1. "The attack machine costs more than this RSA key is worth."

2. "The attack machine isn't off-the-shelf; it's only for attackers building ASICs."

3. For signatures: "We switch keys every month, and the attack machine takes a year."

Real quote: "DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key."

Continuation of qu
despite huge effort
broken a regular 1
fact, the best com
estimated to be th
of a 700-bit key. A
breaking a 1024-bi
would need to exp
amounts of networ
power in a way tha
be detected in ord
single key. Becaus
estimated that mo
safely use 1024-bit
least the next ten

f

ey

key

key

used
keys.
s

Users seem unconcerned:

1. “The attack machine costs more than this RSA key is worth.”

2. “The attack machine isn’t off-the-shelf; it’s only for attackers building ASICs.”

3. For signatures: “We switch keys every month, and the attack machine takes a year.”

Real quote: “DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key.”

Continuation of quote: “To
despite huge efforts, no one
broken a regular 1024-bit ke
fact, the best completed att
estimated to be the equivale
of a 700-bit key. An attacke
breaking a 1024-bit signing
would need to expend phenc
amounts of networked comp
power in a way that would n
be detected in order to brea
single key. Because of this,
estimated that most zones c
safely use 1024-bit keys for a
least the next ten years.”

Users seem unconcerned:

1. "The attack machine costs more than this RSA key is worth."

2. "The attack machine isn't off-the-shelf; it's only for attackers building ASICs."

3. For signatures: "We switch keys every month, and the attack machine takes a year."

Real quote: "DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key."

Continuation of quote: "To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years."

em unconcerned:

attack machine costs
an this RSA key is worth."

attack machine isn't
helf; it's only for
s building ASICs."

ignatures: "We
eys every month, and
ck machine takes a year."

ote: "DNSSEC signing
uld be large enough to
known cryptographic
during the effectivity
f the key."

Continuation of quote: "To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years."

Goal of
analyze
specifica
ratio, of

"Many"

"Price-p
area-tin

"RAM"
bit integ
accessin
realistic;

"Asymp
suppress
speedup

cerned:

achine costs
A key is worth."

achine isn't
nly for
ASICs."

"We
month, and
e takes a year."

SSEC signing
ge enough to
yptographic
effectivity

Continuation of quote: "To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years."

Goal of our "Batc
analyze the *asymp*
specifically *price-p*
ratio, of breaking

"Many": e.g. milli

"Price-performanc
area-time produc

"RAM" metric (ac
bit integers has sa
accessing array of
realistic; "*AT*" me

"Asymptotic": We
suppress polynomi
speedups are supe

sts
vorth.”
’t

d
year.”
ng
to
ic
/

Continuation of quote: “To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years.”

Goal of our “Batch NFS” pa
analyze the *asymptotic* cost
specifically *price-performanc*
ratio, of breaking *many* RSA

“Many”: e.g. millions.

“Price-performance ratio”:
area-time product for chips

“RAM” metric (adding two
bit integers has same cost as
accessing array of size 2^{64}) i
realistic; “*AT*” metric is rea

“Asymptotic”: We systemat
suppress polynomial factors.
speedups are superpolynomi

Continuation of quote: “To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years.”

Goal of our “Batch NFS” paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

“Many”: e.g. millions.

“Price-performance ratio”: **area-time product** for chips.

“RAM” metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; “*AT*” metric is realistic.

“Asymptotic”: We systematically suppress polynomial factors. Our speedups are superpolynomial.

ation of quote: "To date,
huge efforts, no one has
regular 1024-bit key; in
best completed attack is
ed to be the equivalent
-bit key. An attacker
a 1024-bit signing key
eed to expend phenomenal
of networked computing
a way that would not
cted in order to break a
ey. Because of this, it is
ed that most zones can
se 1024-bit keys for at
next ten years."

Goal of our "Batch NFS" paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

"Many": e.g. millions.

"Price-performance ratio": **area-time product** for chips.

"RAM" metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; "AT" metric is realistic.

"Asymptotic": We systematically suppress polynomial factors. Our speedups are superpolynomial.

Best res
time $L^{1.}$
using ch
AT is L

uote: “To date,
ts, no one has
024-bit key; in
pleted attack is
ne equivalent
An attacker
it signing key
pend phenomenal
rked computing
at would not
er to break a
se of this, it is
ost zones can
t keys for at
years.”

Goal of our “Batch NFS” paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

“Many”: e.g. millions.

“Price-performance ratio”: **area-time product** for chips.

“RAM” metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; “AT” metric is realistic.

“Asymptotic”: We systematically suppress polynomial factors. Our speedups are superpolynomial.

Best result known
time $L^{1.185632}$
using chip area L^0
AT is $L^{1.976052}$.

date,
has
y; in
ack is
ent
er
key
omenal
uting
not
k a
it is
can
at

Goal of our "Batch NFS" paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

"Many": e.g. millions.

"Price-performance ratio": **area-time product** for chips.

"RAM" metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; "*AT*" metric is realistic.

"Asymptotic": We systematically suppress polynomial factors. Our speedups are superpolynomial.

Best result known for *one* ke
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Goal of our "Batch NFS" paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

"Many": e.g. millions.

"Price-performance ratio": **area-time product** for chips.

"RAM" metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; "AT" metric is realistic.

"Asymptotic": We systematically suppress polynomial factors. Our speedups are superpolynomial.

Best result known for *one* key: time $L^{1.185632}$ using chip area $L^{0.790420}$; AT is $L^{1.976052}$.

Goal of our "Batch NFS" paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

"Many": e.g. millions.

"Price-performance ratio": **area-time product** for chips.

"RAM" metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; "*AT*" metric is realistic.

"Asymptotic": We systematically suppress polynomial factors. Our speedups are superpolynomial.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

Goal of our "Batch NFS" paper: analyze the *asymptotic* cost, specifically *price-performance ratio*, of breaking *many* RSA keys.

"Many": e.g. millions.

"Price-performance ratio": **area-time product** for chips.

"RAM" metric (adding two 64-bit integers has same cost as accessing array of size 2^{64}) is not realistic; "*AT*" metric is realistic.

"Asymptotic": We systematically suppress polynomial factors. Our speedups are superpolynomial.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

This paper also looks more closely at $L^{o(1)}$, analyzing asymptotic speedup from early-abort ECM. Results are not what one would guess from 1982 Pomerance.

our "Batch NFS" paper:
the *asymptotic* cost,
lly *price-performance*
breaking *many* RSA keys.

: e.g. millions.

erformance ratio":
ne product for chips.

metric (adding two 64-
gers has same cost as
g array of size 2^{64}) is not
"*AT*" metric is realistic.

totic": We systematically
polynomial factors. Our
s are superpolynomial.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

This paper also looks more closely
at $L^{o(1)}$, analyzing asymptotic
speedup from early-abort ECM.
Results are not what one would
guess from 1982 Pomerance.

Asympto

1. Attac
is reduce
can targ

2. Prima
memory
for off-th

3. Attac
(and can
breaking

h NFS" paper:
ptotic cost,
performance
many RSA keys.

ions.

e ratio":
ct for chips.

dding two 64-
me cost as
size 2^{64}) is not
etric is realistic.

e systematically
al factors. Our
rpolynomial.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

This paper also looks more closely
at $L^{o(1)}$, analyzing asymptotic
speedup from early-abort ECM.
Results are not what one would
guess from 1982 Pomerance.

Asymptotic conse

1. Attack cost pe
is reduced, so att
can target lower-v

2. Primary bottle
memory factorizat
for off-the-shelf g

3. Attack time is
(and can be reduc
breaking key rotat

aper:
,
ce
A keys.

s.

64-
s
is not
listic.

tically
Our
al.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

This paper also looks more closely
at $L^{o(1)}$, analyzing asymptotic
speedup from early-abort ECM.
Results are not what one would
guess from 1982 Pomerance.

Asymptotic consequences:

1. Attack cost per key
is reduced, so attacker
can target lower-value keys.

2. Primary bottleneck is low
memory factorization—well
for off-the-shelf graphics car

3. Attack time is reduced
(and can be reduced more),
breaking key rotation.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

This paper also looks more closely at $L^{o(1)}$, analyzing asymptotic speedup from early-abort ECM. Results are not what one would guess from 1982 Pomerance.

Asymptotic consequences:

1. Attack cost per key is reduced, so attacker can target lower-value keys.

2. Primary bottleneck is low-memory factorization—well suited for off-the-shelf graphics cards.

3. Attack time is reduced (and can be reduced more), breaking key rotation.

Best result known for *one* key:
time $L^{1.185632}$
using chip area $L^{0.790420}$;
AT is $L^{1.976052}$.

Our main result for
a batch of $L^{0.5}$ keys:
time $L^{1.022400}$
using chip area $L^{1.181600}$;
AT per key is $L^{1.704000}$.

This paper also looks more closely at $L^{o(1)}$, analyzing asymptotic speedup from early-abort ECM. Results are not what one would guess from 1982 Pomerance.

Asymptotic consequences:

1. Attack cost per key is reduced, so attacker can target lower-value keys.

2. Primary bottleneck is low-memory factorization—well suited for off-the-shelf graphics cards.

3. Attack time is reduced (and can be reduced more), breaking key rotation.

"Do the asymptotics really kick in before 1024 bits?" — Maybe not, but no basis for confidence.

ult known for *one* key:
185632

ip area $L^{0.790420}$;
$^{1.976052}$.

n result for
of $L^{0.5}$ keys:
022400

ip area $L^{1.181600}$;
key is $L^{1.704000}$.

per also looks more closely
, analyzing asymptotic
from early-abort ECM.
are not what one would
om 1982 Pomerance.

Asymptotic consequences:

1. Attack cost per key is reduced, so attacker can target lower-value keys.

2. Primary bottleneck is low-memory factorization—well suited for off-the-shelf graphics cards.

3. Attack time is reduced (and can be reduced more), breaking key rotation.

"Do the asymptotics really kick in before 1024 bits?" — Maybe not, but no basis for confidence.

Eratosth

Sieving s
using pri

1		
2	2	
3		3
4	2 2	
5		
6	2	3
7		
8	2 2 2	
9		3
10	2	
11		
12	2 2	3
13		
14	2	
15		3
16	2 2 2 2	
17		
18	2	3
19		
20	2 2	

etc.

for *one* key:

.790420;

or

ys:

.181600;

704000.

oks more closely

g asymptotic

y-abort ECM.

hat one would

Pomerance.

Asymptotic consequences:

1. Attack cost per key is reduced, so attacker can target lower-value keys.

2. Primary bottleneck is low-memory factorization—well suited for off-the-shelf graphics cards.

3. Attack time is reduced (and can be reduced more), breaking key rotation.

"Do the asymptotics really kick in before 1024 bits?" — Maybe not, but no basis for confidence.

Eratosthenes for s

Sieving small integ

using primes $2, 3, 5$

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

etc.

ey:

closely
tic
CM.
ould
.

Asymptotic consequences:

1. Attack cost per key is reduced, so attacker can target lower-value keys.

2. Primary bottleneck is low-memory factorization—well suited for off-the-shelf graphics cards.

3. Attack time is reduced (and can be reduced more), breaking key rotation.

"Do the asymptotics really kick in before 1024 bits?" — Maybe not, but no basis for confidence.

<u>Eratosthenes for smoothness</u>

Sieving small integers $i > 0$ using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

etc.

Asymptotic consequences:

1. Attack cost per key is reduced, so attacker can target lower-value keys.

2. Primary bottleneck is low-memory factorization—well suited for off-the-shelf graphics cards.

3. Attack time is reduced (and can be reduced more), breaking key rotation.

"Do the asymptotics really kick in before 1024 bits?" — Maybe not, but no basis for confidence.

Eratosthenes for smoothness

Sieving small integers $i > 0$ using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

etc.

otic consequences:

ck cost per key
ed, so attacker
et lower-value keys.

ary bottleneck is low-
 factorization—well suited
he-shelf graphics cards.

ck time is reduced
n be reduced more),
; key rotation.

 asymptotics really kick in
024 bits?" — Maybe not,
basis for confidence.

Eratosthenes for smoothness

Sieving small integers $i > 0$ using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

etc.

The **Q** s

Sieving
using pri

1						
2	2					
3					3	
4	2	2				
5						
6	2				3	
7						
8	2	2	2			
9					3	
10	2					
11						
12	2	2			3	
13						
14	2					
15					3	
16	2	2	2	2		
17						
18	2				3	
19						
20	2	2				

etc.

quences:

r key
ncker
alue keys.

neck is low-
ion—well suited
aphics cards.

reduced
ed more),
ion.

ics really kick in
— Maybe not,
onfidence.

Eratosthenes for smoothness

Sieving small integers $i > 0$
using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

etc.

The **Q** sieve

Sieving i and 611
using primes $2, 3, 5$

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

612	2
613	
614	2
615	
616	2
617	
618	2
619	
620	2
621	
622	2
623	
624	2
625	
626	2
627	
628	2
629	
630	2
631	

etc.

y-
suited
ds.

kick in
e not,

Eratosthenes for smoothness

Sieving small integers $i > 0$ using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

etc.

The **Q** sieve

Sieving i and $611 + i$ for sm
using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

612	2	2			3	3		
613								
614	2							
615					3			5
616	2	2	2					
617								
618	2				3			
619								
620	2	2						5
621					3	3	3	
622	2							
623								
624	2	2	2	2	3			
625								5
626	2							
627					3			
628	2	2						
629								
630	2				3	3		5
631								

etc.

Eratosthenes for smoothness

Sieving small integers $i > 0$ using primes $2, 3, 5, 7$:

i	2	3	5	7
1				
2	2			
3		3		
4	2 2			
5			5	
6	2	3		
7				7
8	2 2 2			
9		3 3		
10	2		5	
11				
12	2 2	3		
13				
14	2			7
15		3	5	
16	2 2 2 2			
17				
18	2	3 3		
19				
20	2 2		5	

etc.

The **Q** sieve

Sieving i and $611 + i$ for small i using primes $2, 3, 5, 7$:

i	2	3	5	7
1				
2	2			
3		3		
4	2 2			
5			5	
6	2	3		
7				7
8	2 2 2			
9		3 3		
10	2		5	
11				
12	2 2	3		
13				
14	2			7
15		3	5	
16	2 2 2 2			
17				
18	2	3 3		
19				
20	2 2		5	

$611+i$	2	3	5	7
612	2 2	3 3		
613				
614	2			
615		3	5	
616	2 2 2			7
617				
618	2	3		
619				
620	2 2		5	
621		3 3 3		
622	2			
623				7
624	2 2 2 2	3		
625			5 5 5 5	
626	2			
627		3		
628	2 2			
629				
630	2	3 3	5	7
631				

etc.

nenes for smoothness

small integers $i > 0$

imes $2, 3, 5, 7$:

	5	
		7
3		
	5	
		7
	5	
3		
	5	

The $\mathbf{Q}$ sieve

Sieving i and $611 + i$ for small i using primes $2, 3, 5, 7$:

1								
2	2							
3					3			
4	2	2						
5							5	
6	2				3			
7								7
8	2	2	2					
9					3	3		
10	2						5	
11								
12	2	2			3			
13								
14	2							7
15					3		5	
16	2	2	2	2				
17								
18	2				3	3		
19								
20	2	2					5	

612	2	2			3	3						
613												
614	2											
615					3			5				
616	2	2	2									7
617												
618	2				3							
619												
620	2	2						5				
621					3	3	3					
622	2											
623												7
624	2	2	2	2	3							
625								5	5	5	5	
626	2											
627					3							
628	2	2										
629												
630	2				3	3		5				7
631												

etc.

Have co

the cong

for some

$14 \cdot 625$

$64 \cdot 675$

$75 \cdot 686$

$14 \cdot 64 \cdot$

$= 2^8 3^4 5$

$\gcd\{611$

$= 47.$

$611 = 4$

moothness

gers $i > 0$

$5, 7$:

The **Q** sieve

Sieving i and $611 + i$ for small i using primes $2, 3, 5, 7$:

1	
2	2
3	3
4	2 2
5	5
6	2 3
7	7
8	2 2 2
9	3 3
10	2 5
11	
12	2 2 3
13	
14	2 7
15	3 5
16	2 2 2 2
17	
18	2 3 3
19	
20	2 2 5

612	2 2 3 3
613	
614	2
615	3 5
616	2 2 2 7
617	
618	2 3
619	
620	2 2 5
621	3 3 3
622	2
623	7
624	2 2 2 2 3
625	5 5 5 5
626	2
627	3
628	2 2
629	
630	2 3 3 5 7
631	

etc.

Have complete fac

the congruences i

for some i's.

$14 \cdot 625 = 2^1 3^0 5^4$

$64 \cdot 675 = 2^6 3^3 5^2$

$75 \cdot 686 = 2^1 3^1 5^2$

$14 \cdot 64 \cdot 75 \cdot 625 \cdot$

$= 2^8 3^4 5^8 7^4 = (2^4$

$\gcd\{611, 14 \cdot 64 \cdot$

$= 47.$

$611 = 47 \cdot 13.$

s

The **Q** sieve

Sieving i and $611 + i$ for small i using primes $2, 3, 5, 7$:

1				
2	2			
3		3		
4	2 2			
5			5	
6	2	3		
7				7
8	2 2 2			
9		3 3		
10	2		5	
11				
12	2 2	3		
13				
14	2			7
15		3	5	
16	2 2 2 2			
17				
18	2	3 3		
19				
20	2 2		5	

612	2 2	3 3		
613				
614	2			
615		3	5	
616	2 2 2			7
617				
618	2	3		
619				
620	2 2		5	
621		3 3 3		
622	2			
623				7
624	2 2 2 2	3		
625			5 5 5 5	
626	2			
627		3		
628	2 2			
629				
630	2	3 3	5	7
631				

etc.

Have complete factorization
the congruences $i \equiv 611 + i$
for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5$
$= 47$.

$611 = 47 \cdot 13$.

The **Q** sieve

Sieving i and $611 + i$ for small i using primes $2, 3, 5, 7$:

i	2	3	5	7
1				
2	2			
3		3		
4	2 2			
5			5	
6	2	3		
7				7
8	2 2 2			
9		3 3		
10	2		5	
11				
12	2 2	3		
13				
14	2			7
15		3	5	
16	2 2 2 2			
17				
18	2	3 3		
19				
20	2 2		5	

$611+i$	2	3	5	7
612	2 2	3 3		
613				
614	2			
615		3	5	
616	2 2 2			7
617				
618	2	3		
619				
620	2 2		5	
621		3 3 3		
622	2			
623				7
624	2 2 2 2	3		
625			5 5 5 5	
626	2			
627		3		
628	2 2			
629				
630	2	3 3	5	7
631				

etc.

Have complete factorization of the congruences $i \equiv 611 + i$ for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$
$= 47$.

$611 = 47 \cdot 13$.

sieve

i and $611 + i$ for small i

imes $2, 3, 5, 7$:

	5	
		7
3		
	5	
		7
	5	
3		
	5	

612	2	2			3	3						
613												
614	2											
615					3			5				
616	2	2	2									7
617												
618	2				3							
619												
620	2	2						5				
621					3	3	3					
622	2											
623												7
624	2	2	2	2	3							
625								5	5	5	5	
626	2											
627					3							
628	2	2										
629												
630	2				3	3		5				7
631												

Have complete factorization of the congruences $i \equiv 611 + i$ for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$
$= 47$.

$611 = 47 \cdot 13$.

The nun

Generali

$\rightarrow a \equiv a$

$\rightarrow a - b$

for root

of nonze

For any

so that f

produces

Optimal

$(\mu + o(1$

$+ i$ for small i

$5, 7$:

2	3 3		
	3	5	
2 2			7
	3		
2		5	
	3 3 3		
			7
2 2 2	3		
		5 5 5 5	
	3		
2			
	3 3	5	7

Have complete factorization of the congruences $i \equiv 611 + i$ for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$
$= 47$.

$611 = 47 \cdot 13$.

The number-field s

Generalize $i \equiv i +$
$\to a \equiv a + bN \quad ($
$\to a - bm \equiv a - b$
for root $\alpha \in \mathbf{C}$
of nonzero integer

For any m can fin
so that factoring n
produces factoriza

Optimal choice of
$(\mu + o(1))(\log N)^2$

all i

7

7

5 5 5

7

Have complete factorization of the congruences $i \equiv 611 + i$ for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$
$= 47$.

$611 = 47 \cdot 13$.

The number-field sieve

Generalize $i \equiv i + N \pmod$
$\to a \equiv a + bN \pmod N$
$\to a - bm \equiv a - b\alpha \pmod$
for root $\alpha \in \mathbf{C}$
of nonzero integer poly.

For any m can find α
so that factoring $m - \alpha$
produces factorization of N.

Optimal choice of $\log m$ is
$(\mu + o(1))(\log N)^{2/3}(\log\log$

Have complete factorization of the congruences $i \equiv 611 + i$ for some i's.

$14 \cdot 625 = 2^1 3^0 5^4 7^1$.
$64 \cdot 675 = 2^6 3^3 5^2 7^0$.
$75 \cdot 686 = 2^1 3^1 5^2 7^3$.

$14 \cdot 64 \cdot 75 \cdot 625 \cdot 675 \cdot 686$
$= 2^8 3^4 5^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$\gcd\{611, 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$
$= 47$.

$611 = 47 \cdot 13$.

The number-field sieve

Generalize $i \equiv i + N \pmod N$
$\to a \equiv a + bN \pmod N$
$\to a - bm \equiv a - b\alpha \pmod{m - \alpha}$
for root $\alpha \in \mathbf{C}$
of nonzero integer poly.

For any m can find α
so that factoring $m - \alpha$
produces factorization of N.

Optimal choice of $\log m$ is
$(\mu + o(1))(\log N)^{2/3}(\log\log N)^{1/3}$.

mplete factorization of
gruences $i \equiv 611 + i$
e i's.

$= 2^1 3^0 5^4 7^1$.
$= 2^6 3^3 5^2 7^0$.
$= 2^1 3^1 5^2 7^3$.

$75 \cdot 625 \cdot 675 \cdot 686$
${}^8 7^4 = (2^4 3^2 5^4 7^2)^2$.

$., 14 \cdot 64 \cdot 75 - 2^4 3^2 5^4 7^2\}$

$7 \cdot 13$.

The number-field sieve

Generalize $i \equiv i + N \pmod N$
$\rightarrow a \equiv a + bN \pmod N$
$\rightarrow a - bm \equiv a - b\alpha \pmod{m - \alpha}$
for root $\alpha \in \mathbf{C}$
of nonzero integer poly.

For any m can find α
so that factoring $m - \alpha$
produces factorization of N.

Optimal choice of $\log m$ is
$(\mu + o(1))(\log N)^{2/3}(\log\log N)^{1/3}$.

RAM co

1993 Bu
Smoothn
Sieve L^1
Find $L^{0.}$
with $a -$
Total RA

1993 Co
Total RA
using mu

(Multipl
don't see
with AT

ctorization of

$\equiv 611 + i$

7^1.

7^0.

7^3.

$675 \cdot 686$

$3^2 5^4 7^2)^2$.

$75 - 2^4 3^2 5^4 7^2\}$

The number-field sieve

Generalize $i \equiv i + N \pmod{N}$

$\rightarrow a \equiv a + bN \pmod{N}$

$\rightarrow a - bm \equiv a - b\alpha \pmod{m - \alpha}$

for root $\alpha \in \mathbf{C}$

of nonzero integer poly.

For any m can find α

so that factoring $m - \alpha$

produces factorization of N.

Optimal choice of $\log m$ is

$(\mu + o(1))(\log N)^{2/3}(\log\log N)^{1/3}$.

RAM cost analysis

1993 Buhler–Lenst

Smoothness bounc

Sieve $L^{1.923000}$ pai

Find $L^{0.961500}$ pair

with $a - bm$ and a

Total RAM time L

1993 Coppersmith

Total RAM time L

using multiple num

(Multiple number

don't seem to com

with AT, factory,

of

$5^4 7^2\}$

The number-field sieve

Generalize $i \equiv i + N \pmod{N}$
$\rightarrow a \equiv a + bN \pmod{N}$
$\rightarrow a - bm \equiv a - b\alpha \pmod{m - \alpha}$
for root $\alpha \in \mathbf{C}$
of nonzero integer poly.

For any m can find α
so that factoring $m - \alpha$
produces factorization of N.

Optimal choice of $\log m$ is
$(\mu + o(1))(\log N)^{2/3}(\log \log N)^{1/3}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomer
Smoothness bound $L^{0.96150}$
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ sm
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

The number-field sieve

Generalize $i \equiv i + N \pmod{N}$
$\rightarrow a \equiv a + bN \pmod{N}$
$\rightarrow a - bm \equiv a - b\alpha \pmod{m - \alpha}$
for root $\alpha \in \mathbf{C}$
of nonzero integer poly.

For any m can find α
so that factoring $m - \alpha$
produces factorization of N.

Optimal choice of $\log m$ is
$(\mu + o(1))(\log N)^{2/3}(\log\log N)^{1/3}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

mber-field sieve

ze $i \equiv i + N \pmod{N}$
$a + bN \pmod{N}$
$m \equiv a - b\alpha \pmod{m - \alpha}$
$\alpha \in \mathbf{C}$
ero integer poly.

m can find α
factoring $m - \alpha$
s factorization of N.

choice of $\log m$ is
$))(\log N)^{2/3}(\log\log N)^{1/3}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost

Sieving i
in realist
AT cost

sieve

$N \pmod{N}$
mod N)
$\alpha \pmod{m-\alpha}$

poly.

d α
$n - \alpha$
tion of N.

log m is
$^{2/3}(\log\log N)^{1/3}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost analysis

Sieving is a disaste
in realistic cost me
AT cost $L^{2.403750}$.

$N)$

$m-\alpha)$

$N)^{1/3}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

Linear algebra is also a disaster.
AT cost $L^{2.403750}$.

RAM cost analysis

1993 Buhler–Lenstra–Pomerance:
Smoothness bound $L^{0.961500}$.
Sieve $L^{1.923000}$ pairs (a, b).
Find $L^{0.961500}$ pairs
with $a - bm$ and $a - b\alpha$ smooth.
Total RAM time $L^{1.923000}$.

1993 Coppersmith:
Total RAM time $L^{1.901884}$
using multiple number fields.

(Multiple number fields
don't seem to combine well
with AT, factory, et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

Linear algebra is also a disaster.
AT cost $L^{2.403750}$.

Semi-fix: Reduce smoothness
bounds to rebalance.
AT cost $L^{1.976052}$.
(2001 Bernstein)

st analysis

hler–Lenstra–Pomerance:
ness bound $L^{0.961500}$.
$.923000$ pairs (a, b).
961500 pairs
$- bm$ and $a - b\alpha$ smooth.
AM time $L^{1.923000}$.

ppersmith:
AM time $L^{1.901884}$
ultiple number fields.

e number fields
em to combine well
, factory, et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

Linear algebra is also a disaster.
AT cost $L^{2.403750}$.

Semi-fix: Reduce smoothness
bounds to rebalance.
AT cost $L^{1.976052}$.
(2001 Bernstein)

The fact

1993 Co
There ex
that fact
with san
in RAM

Smoothn
Smaller
so need

Algorith
such tha
Note: or
Algorith
whether

s

tra–Pomerance:
d $L^{0.961500}$.
irs (a, b).
rs
$a - b\alpha$ smooth.
$L^{1.923000}$.

:
$L^{1.901884}$

mber fields.

fields
nbine well
et al.)

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

Linear algebra is also a disaster.
AT cost $L^{2.403750}$.

Semi-fix: Reduce smoothness
bounds to rebalance.
AT cost $L^{1.976052}$.
(2001 Bernstein)

The factorization

1993 Coppersmith
There *exists* an al
that factors any in
with same #bits a
in RAM time $L^{1.63}$

Smoothness bound
Smaller than befor
so need more $(a, b$

Algorithm *knows* a
such that $a - bm$
Note: one m work
Algorithm uses EC
whether $a - b\alpha_N$

rance:
0.

nooth.

s.

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

Linear algebra is also a disaster.
AT cost $L^{2.403750}$.

Semi-fix: Reduce smoothness
bounds to rebalance.
AT cost $L^{1.976052}$.
(2001 Bernstein)

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.81929}$
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth
Note: one m works for all N
Algorithm uses ECM to chec
whether $a - b\alpha_N$ is smooth.

AT cost analysis

Sieving is a disaster
in realistic cost metric.
AT cost $L^{2.403750}$.

Fix: find smooth using ECM.
AT cost $L^{1.923000}$.

Linear algebra is also a disaster.
AT cost $L^{2.403750}$.

Semi-fix: Reduce smoothness
bounds to rebalance.
AT cost $L^{1.976052}$.
(2001 Bernstein)

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

analysis

is a disaster
tic cost metric.
$L^{2.403750}$.

d smooth using ECM.
$L^{1.923000}$.

lgebra is also a disaster.
$L^{2.403750}$.

: Reduce smoothness
to rebalance.
$L^{1.976052}$.

ernstein)

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

Finding
is slower
Need to
such tha
RAM tir

er

etric.

using ECM.

lso a disaster.

smoothness

ce.

<u>The factorization factory</u>

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

Finding this algori
is slower than runn
Need to precompu
such that $a - bm$
RAM time $L^{2.0068}$

M.

ster.

ss

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

Finding this algorithm
is slower than running it.
Need to precompute all $(a, b$
such that $a - bm$ is smooth
RAM time $L^{2.006853}$.

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

Standard conversion of
precomputation into batching:
if there are enough targets,
more than $L^{0.368266}$,
then precomputation cost
becomes negligible.

The factorization factory

1993 Coppersmith:
There *exists* an algorithm
that factors any integer
with same #bits as N
in RAM time $L^{1.638587}$.

Smoothness bound $L^{0.819290}$.
Smaller than before,
so need more (a, b).

Algorithm *knows* all (a, b)
such that $a - bm$ is smooth.
Note: one m works for all N.
Algorithm uses ECM to check
whether $a - b\alpha_N$ is smooth.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

Standard conversion of
precomputation into batching:
if there are enough targets,
more than $L^{0.368266}$,
then precomputation cost
becomes negligible.

The big problem: Coppersmith's
algorithm has size $L^{1.638587}$.
Huge AT cost; useless in reality.

torization factory

ppersmith:
xists an algorithm
tors any integer
ne #bits as N
time $L^{1.638587}$.

ness bound $L^{0.819290}$.
than before,
more (a, b).

m *knows* all (a, b)
t $a - bm$ is smooth.
ne m works for all N.
m uses ECM to check
$a - b\alpha_N$ is smooth.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

Standard conversion of
precomputation into batching:
if there are enough targets,
more than $L^{0.368266}$,
then precomputation cost
becomes negligible.

The big problem: Coppersmith's
algorithm has size $L^{1.638587}$.
Huge *AT* cost; useless in reality.

Batch N

Goal: O

1. Gene
Test $a -$

2. Make
close to
When sr
test each

3. After
reorgani
relevant

4. Linea

factory

:

gorithm

nteger

ns N

38587.

d $L^{0.819290}$.

re,

b).

all (a, b)

is smooth.

ks for all N.

CM to check

is smooth.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

Standard conversion of
precomputation into batching:
if there are enough targets,
more than $L^{0.368266}$,
then precomputation cost
becomes negligible.

The big problem: Coppersmith's
algorithm has size $L^{1.638587}$.
Huge AT cost; useless in reality.

Batch NFS

Goal: Optimize A

1. Generate (a, b)
Test $a - bm$ for sr

2. Make many cop
close to each $(a, b$
When smooth $a -$
test each $a - b\alpha_N$

3. After all smoot
reorganize: for eac
relevant (a, b) clos

4. Linear algebra.

0 .

.

l.

ck

.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

Standard conversion of
precomputation into batching:
if there are enough targets,
more than $L^{0.368266}$,
then precomputation cost
becomes negligible.

The big problem: Coppersmith's
algorithm has size $L^{1.638587}$.
Huge AT cost; useless in reality.

Batch NFS

Goal: Optimize AT asympto

1. Generate (a, b) in paralle
Test $a - bm$ for smoothness

2. Make many copies of eac
close to each (a, b) generato
When smooth $a - bm$ is fou
test each $a - b\alpha_N$ for smoot

3. After all smooths are fou
reorganize: for each N, brin
relevant (a, b) close togethe

4. Linear algebra.

Finding this algorithm
is slower than running it.
Need to precompute all (a, b)
such that $a - bm$ is smooth.
RAM time $L^{2.006853}$.

Standard conversion of
precomputation into batching:
if there are enough targets,
more than $L^{0.368266}$,
then precomputation cost
becomes negligible.

The big problem: Coppersmith's
algorithm has size $L^{1.638587}$.
Huge AT cost; useless in reality.

Batch NFS

Goal: Optimize AT asymptotics.

1. Generate (a, b) in parallel.
Test $a - bm$ for smoothness.

2. Make many copies of each N,
close to each (a, b) generator.
When smooth $a - bm$ is found,
test each $a - b\alpha_N$ for smoothness.

3. After all smooths are found,
reorganize: for each N, bring
relevant (a, b) close together.

4. Linear algebra.

this algorithm
r than running it.
precompute all (a, b)
nt $a - bm$ is smooth.
me $L^{2.006853}$.

d conversion of
putation into batching:
are enough targets,
an $L^{0.368266}$,
computation cost
s negligible.

problem: Coppersmith's
m has size $L^{1.638587}$.
T cost; useless in reality.

Batch NFS

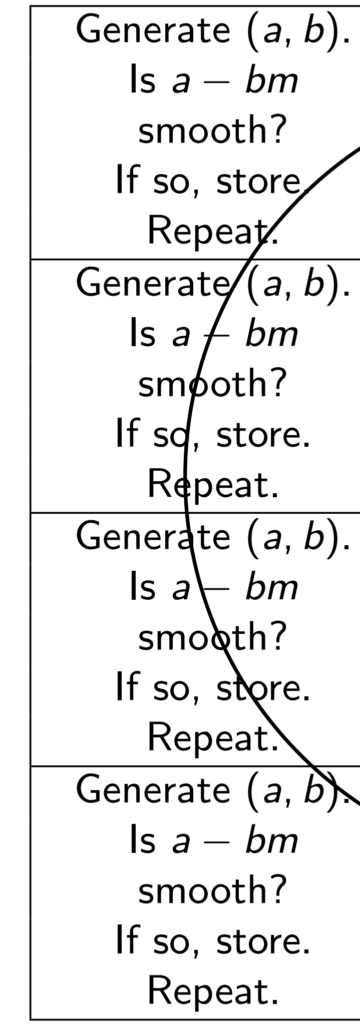
Goal: Optimize AT asymptotics.

1. Generate (a, b) in parallel.
Test $a - bm$ for smoothness.

2. Make many copies of each N,
close to each (a, b) generator.
When smooth $a - bm$ is found,
test each $a - b\alpha_N$ for smoothness.

3. After all smooths are found,
reorganize: for each N, bring
relevant (a, b) close together.

4. Linear algebra.



thm
hing it.
te all (a, b)
is smooth.
^{53}.

on of
to batching:
n targets,
^{66},
on cost
e.

Coppersmith's
$L^{1.638587}$.
eless in reality.

Batch NFS

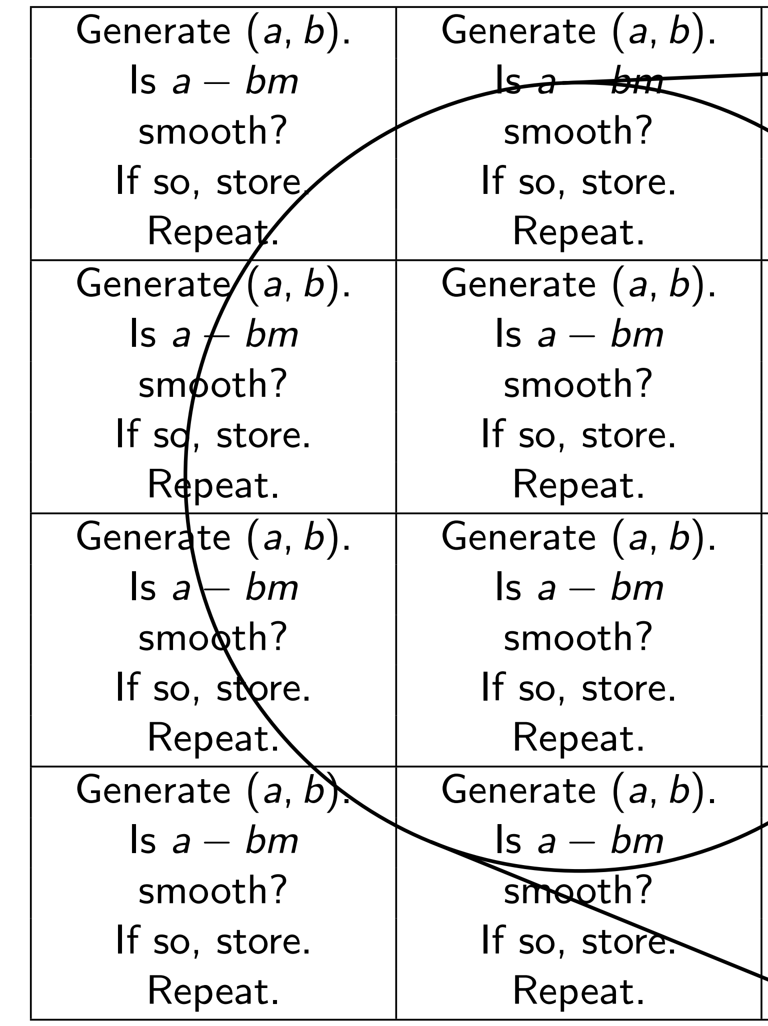
Goal: Optimize AT asymptotics.

1. Generate (a, b) in parallel. Test $a - bm$ for smoothness.

2. Make many copies of each N, close to each (a, b) generator. When smooth $a - bm$ is found, test each $a - b\alpha_N$ for smoothness.

3. After all smooths are found, reorganize: for each N, bring relevant (a, b) close together.

4. Linear algebra.



b)

.

g:

ith's

ality.

Batch NFS

Goal: Optimize AT asymptotics.

1. Generate (a, b) in parallel.
Test $a - bm$ for smoothness.

2. Make many copies of each N, close to each (a, b) generator.
When smooth $a - bm$ is found, test each $a - b\alpha_N$ for smoothness.

3. After all smooths are found, reorganize: for each N, bring relevant (a, b) close together.

4. Linear algebra.

Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.

Batch NFS

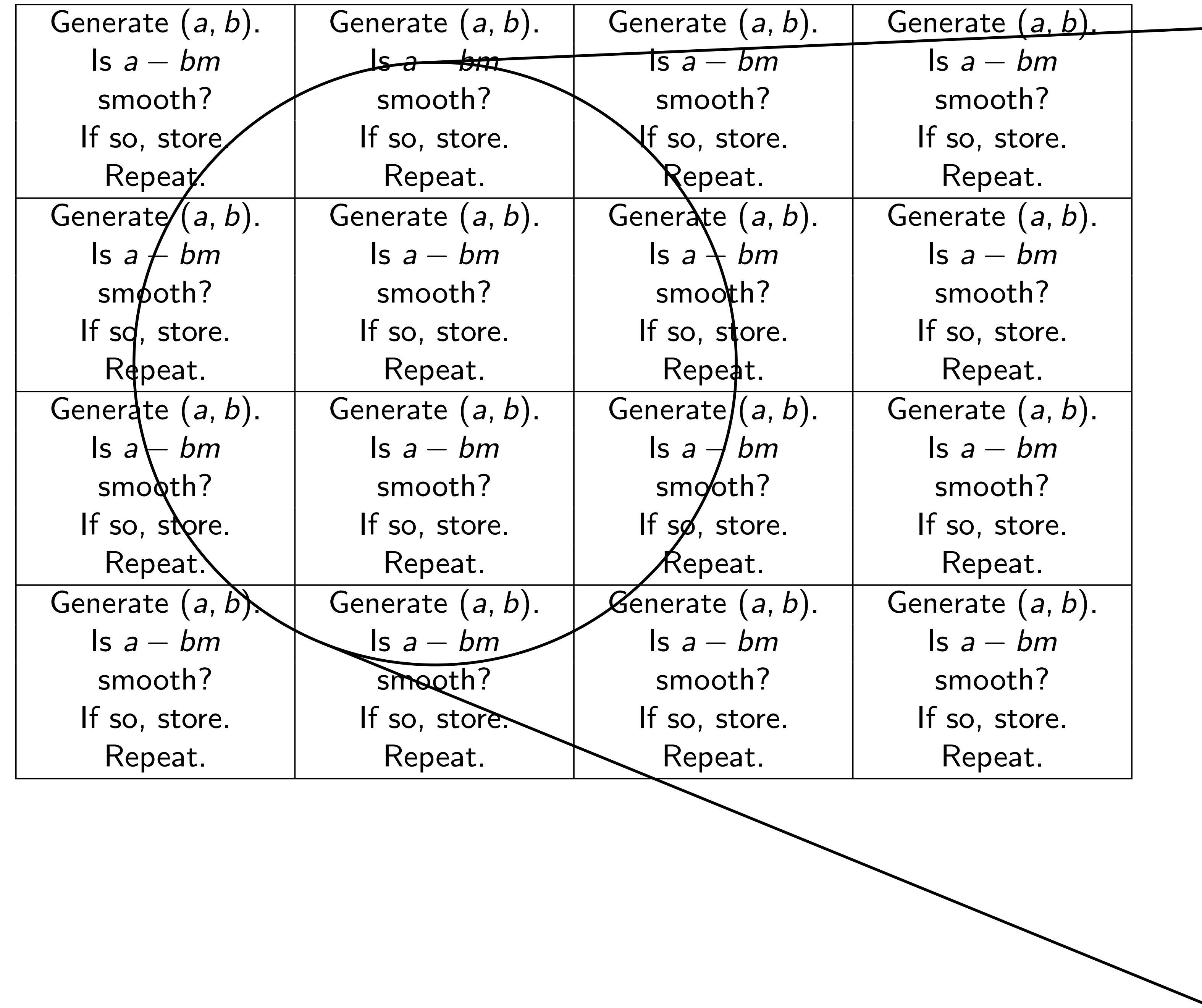
Goal: Optimize AT asymptotics.

1. Generate (a, b) in parallel. Test $a - bm$ for smoothness.

2. Make many copies of each N, close to each (a, b) generator. When smooth $a - bm$ is found, test each $a - b\alpha_N$ for smoothness.

3. After all smooths are found, reorganize: for each N, bring relevant (a, b) close together.

4. Linear algebra.



FS

ptimize AT asymptotics.

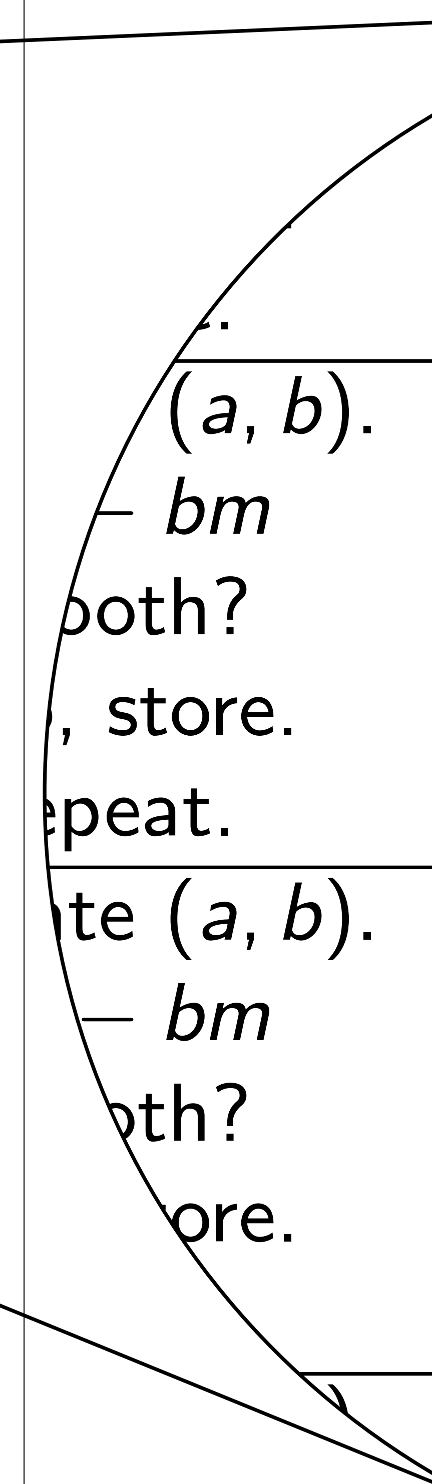
rate (a, b) in parallel.
$- bm$ for smoothness.

many copies of each N,
each (a, b) generator.
mooth $a - bm$ is found,
$a - b\alpha_N$ for smoothness.

all smooths are found,
ze: for each N, bring
(a, b) close together.

r algebra.

Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.



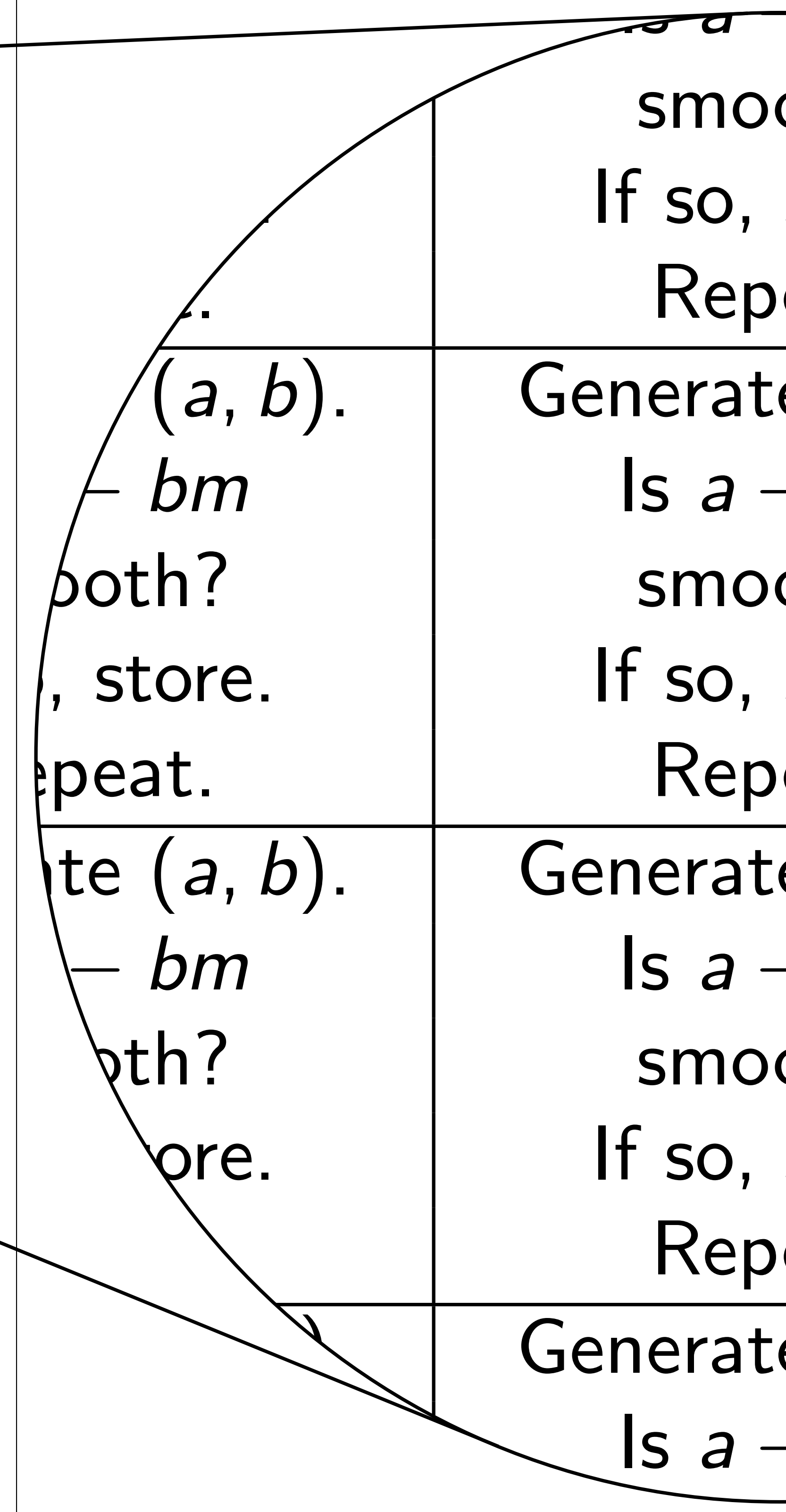
T asymptotics.

in parallel.
moothness.

pies of each N,
) generator.
bm is found,
for smoothness.

hs are found,
ch N, bring
se together.

Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.



otics.

l.

s.

th N,

or.

und,

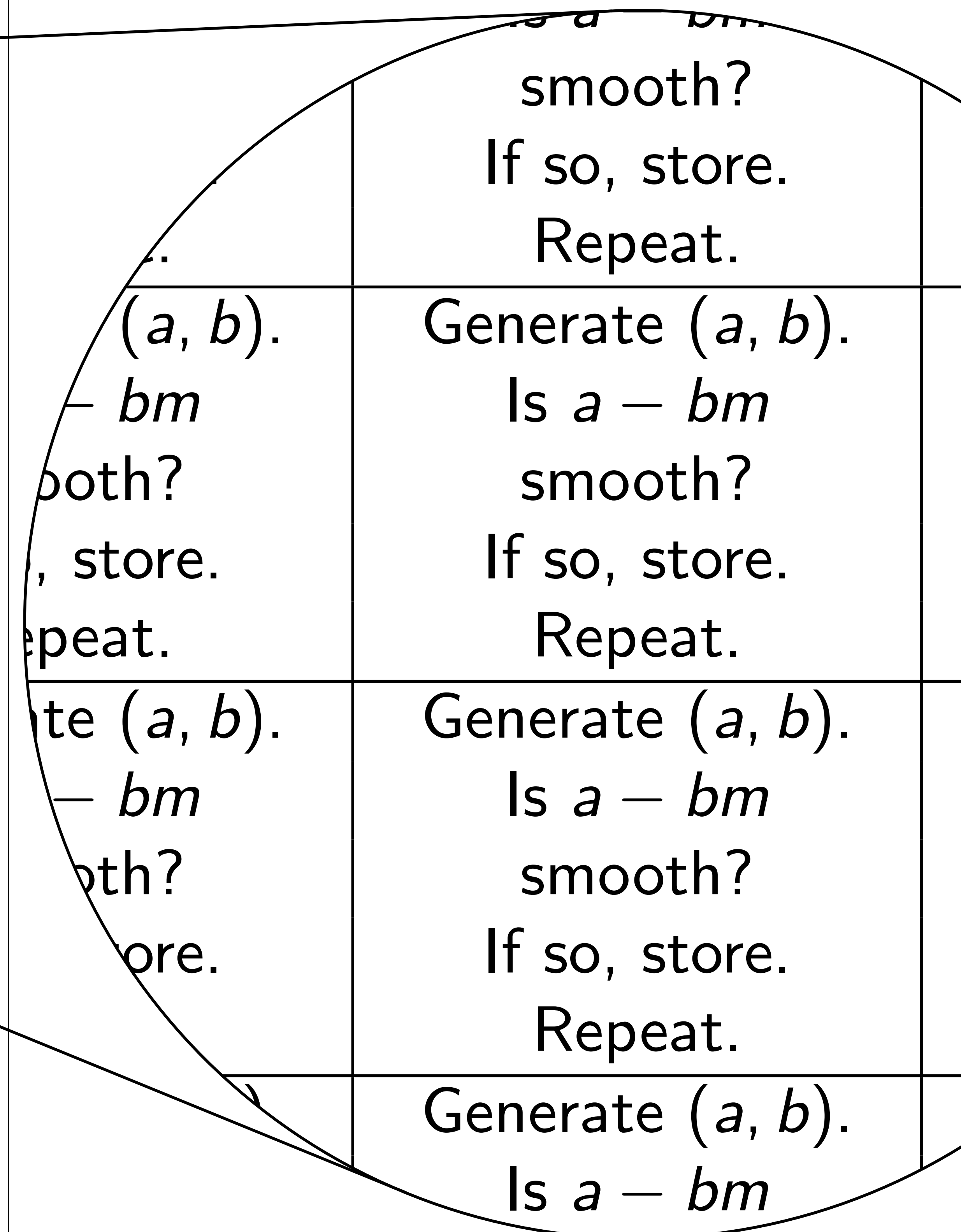
thness.

nd,

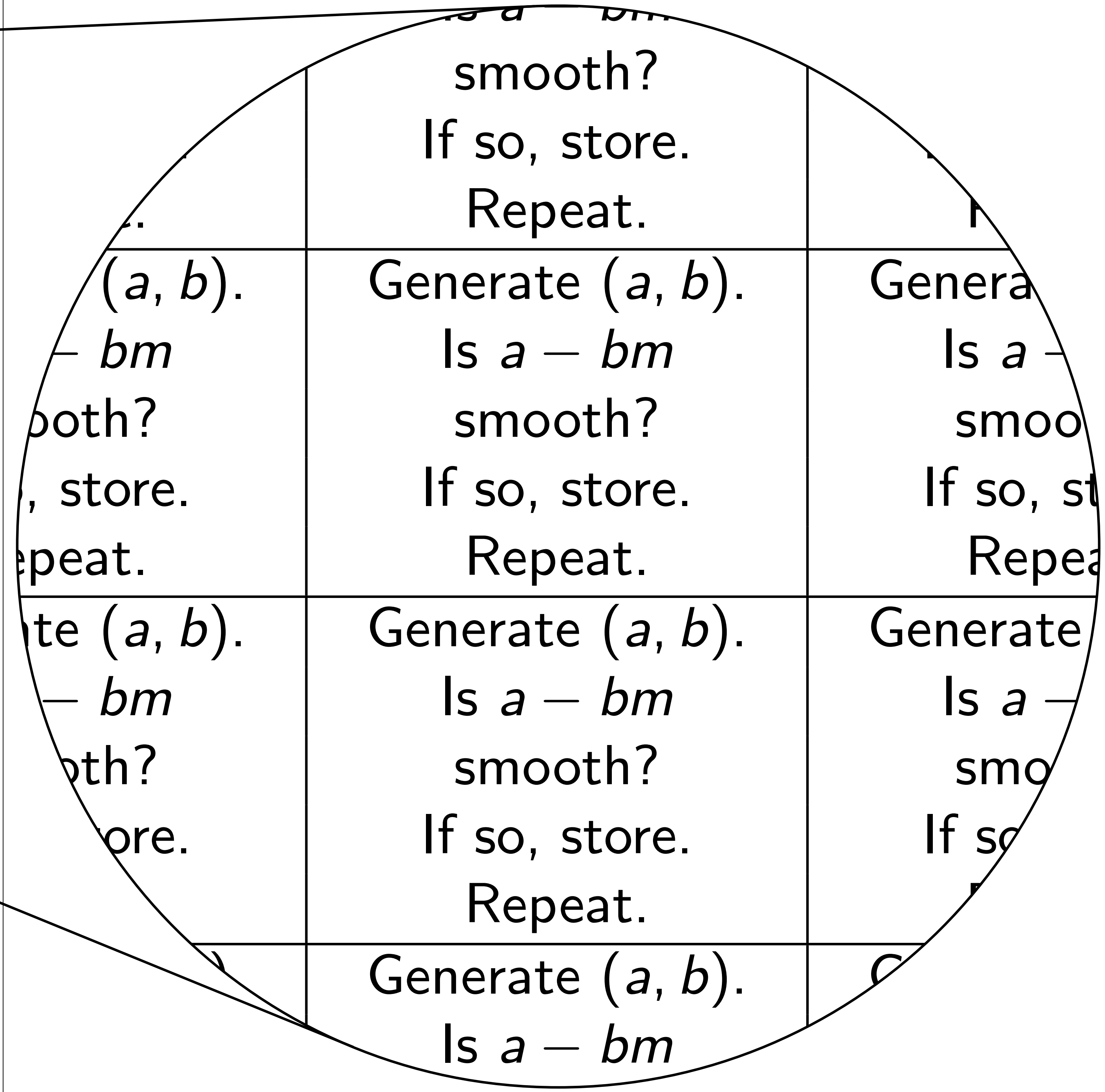
g

r.

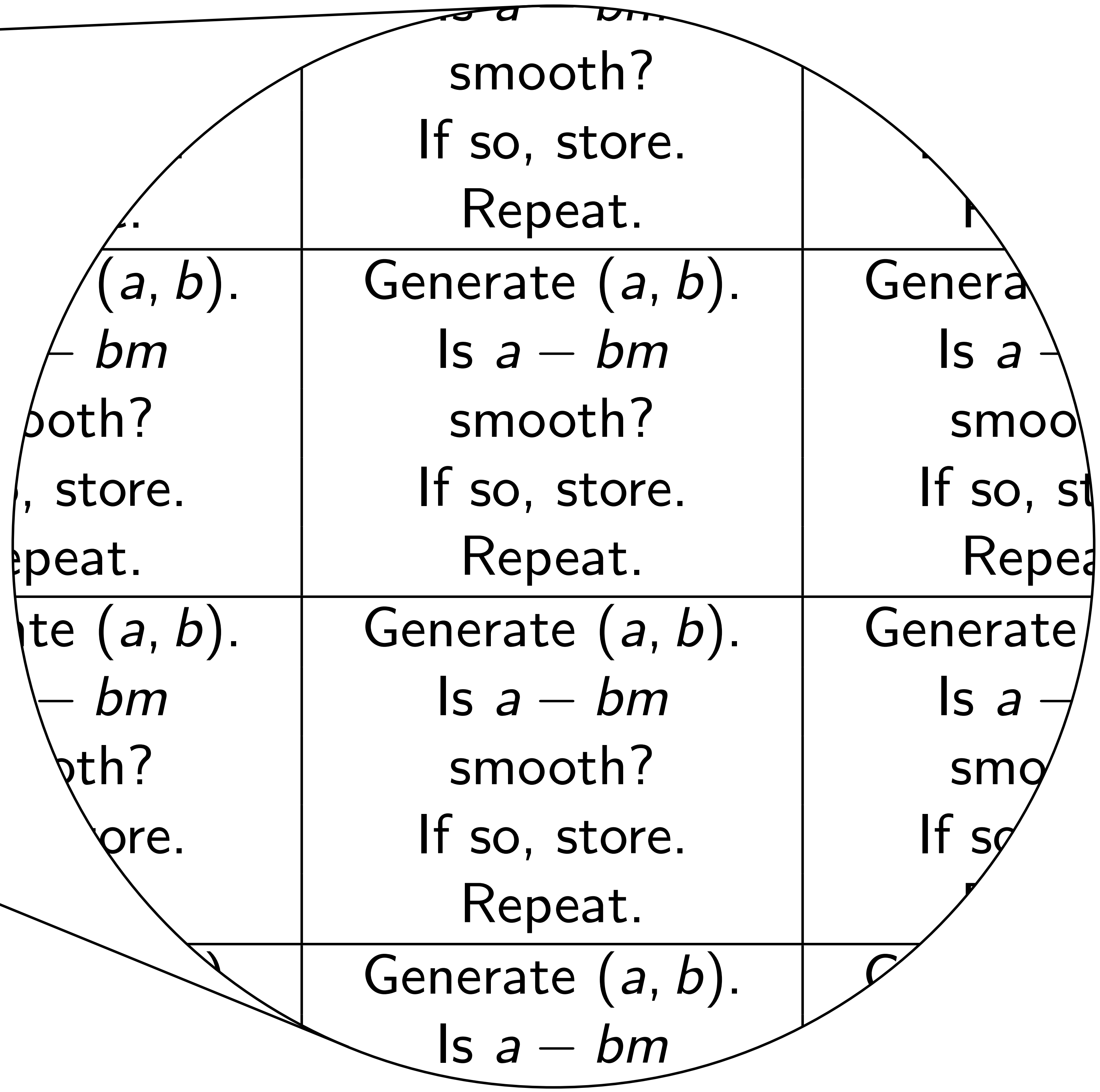
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.



Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.

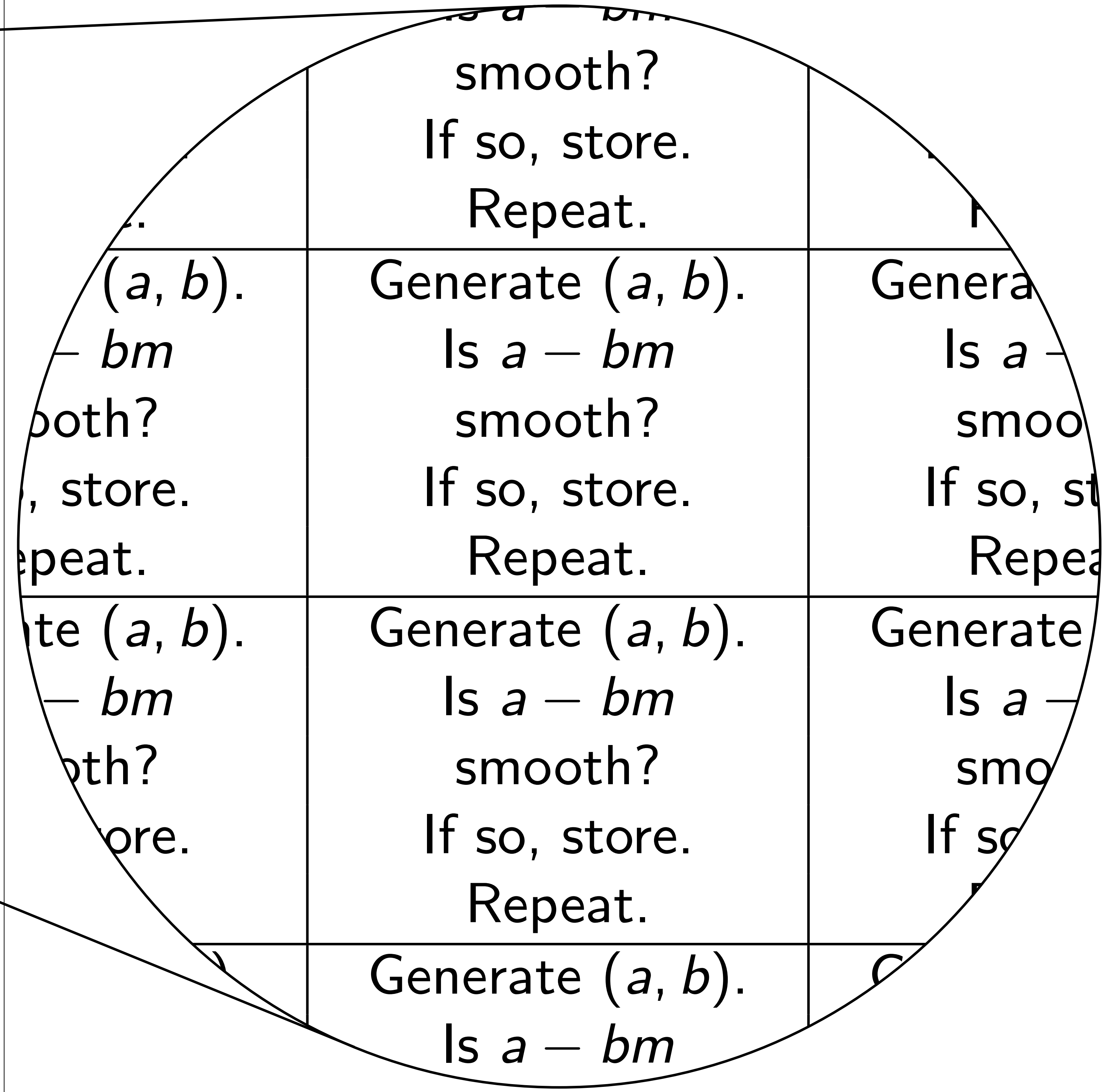


Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.

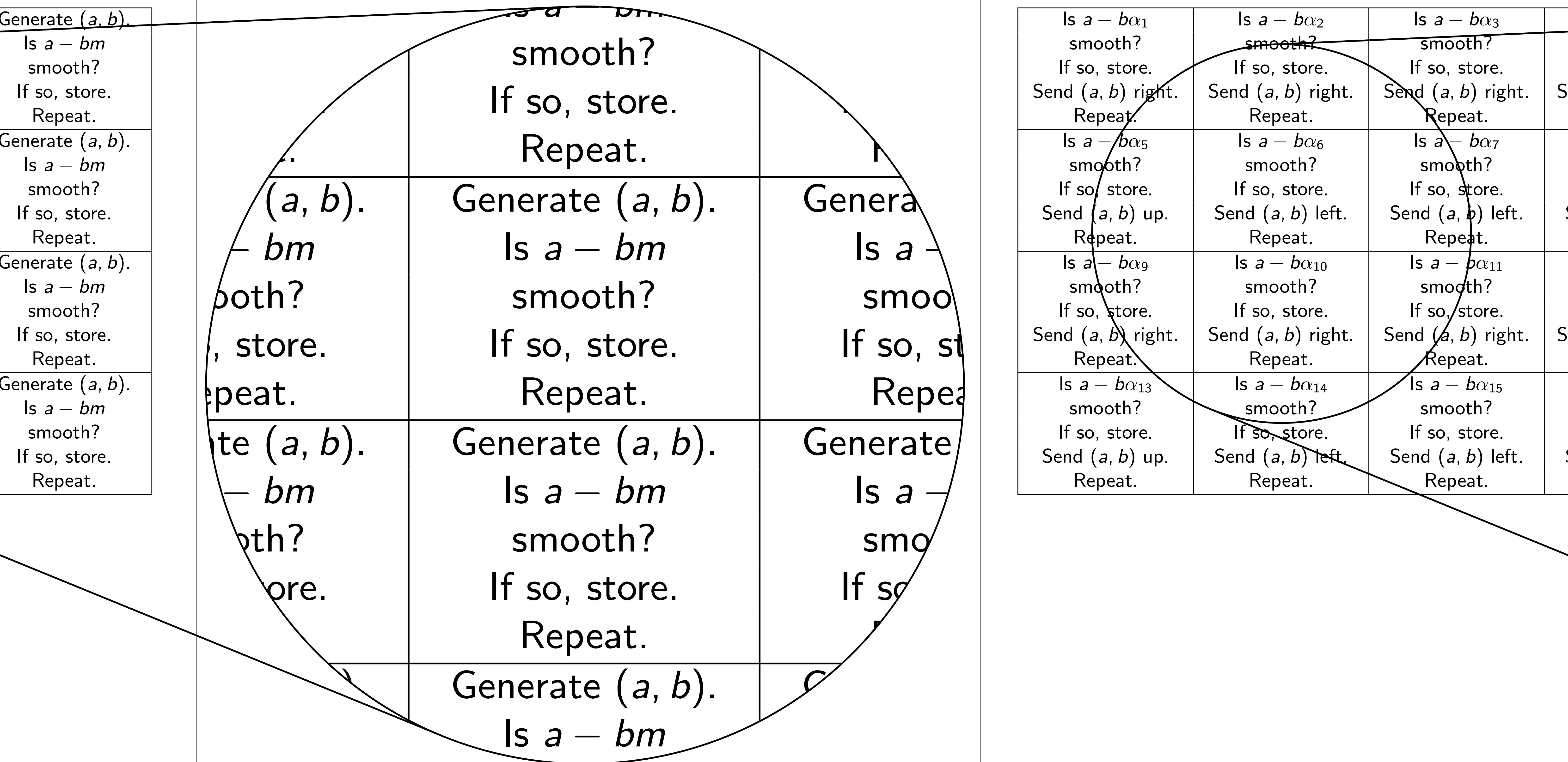


Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.

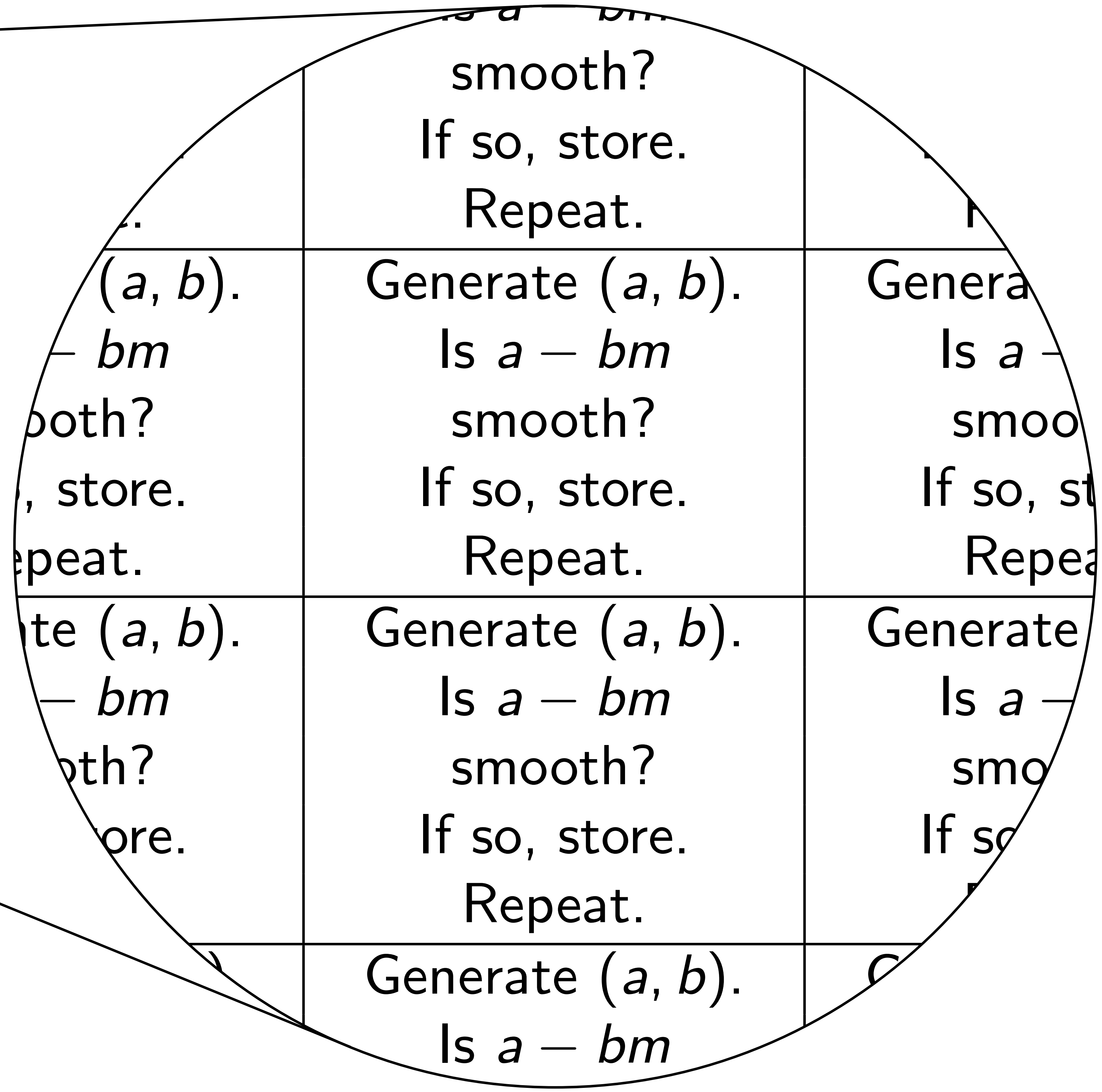
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.
Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.	Generate (a, b). Is $a - bm$ smooth? If so, store. Repeat.



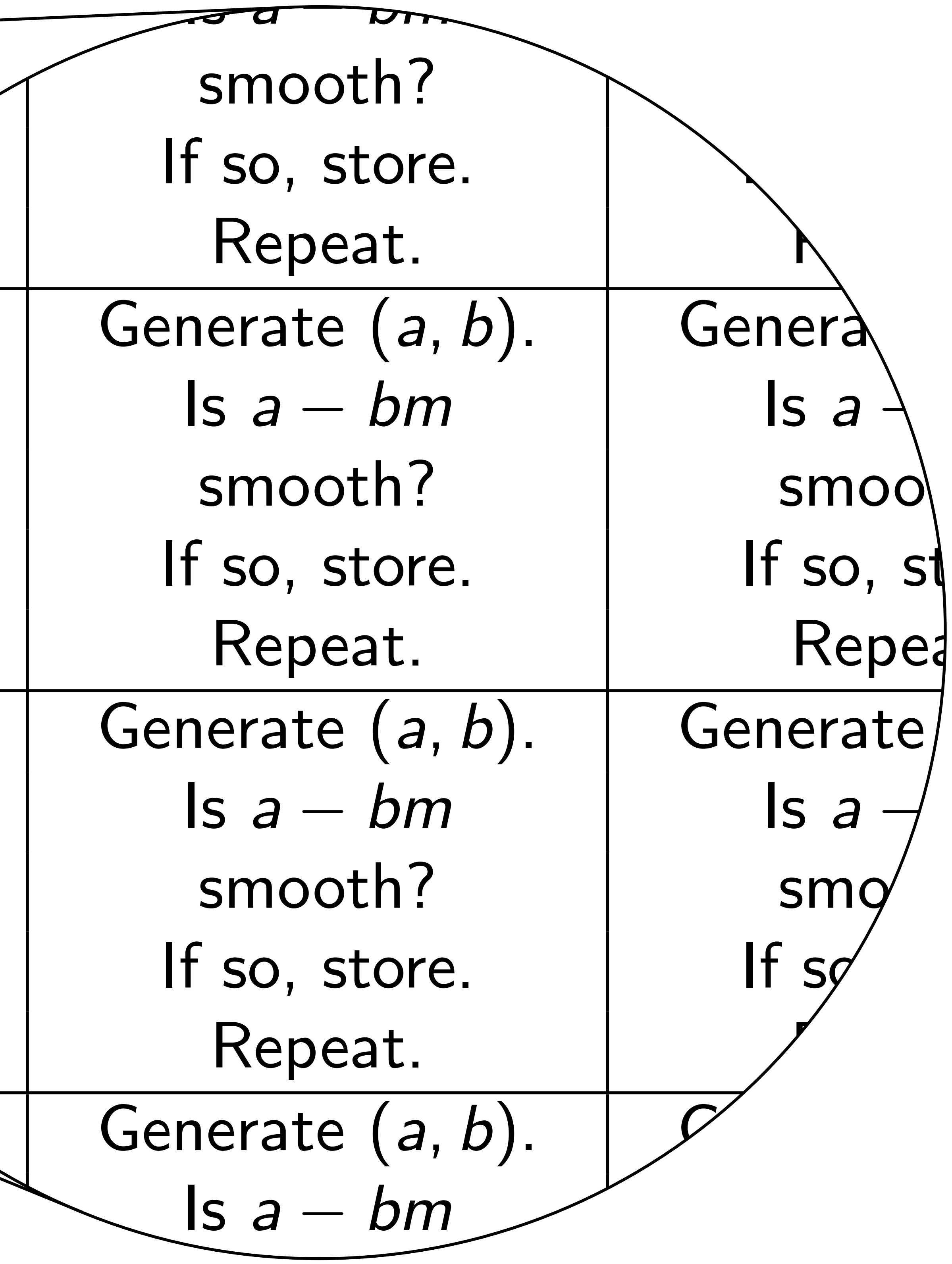
Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.



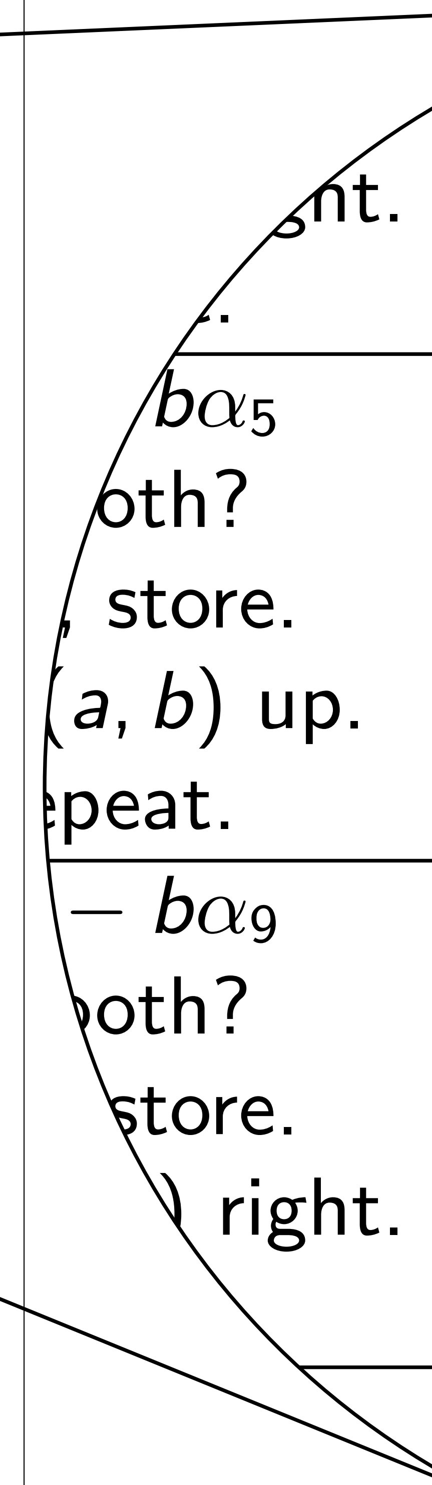
Generate (a, b).
Is $a - bm$
smooth?
If so, store.
Repeat.
Is $a - b\alpha_1$
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is $a - b\alpha_2$
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is $a - b\alpha_3$
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is $a - b\alpha_5$
smooth?
If so, store.
Send (a, b) up.
Repeat.
Is $a - b\alpha_6$
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is $a - b\alpha_7$
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is $a - b\alpha_9$
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is $a - b\alpha_{10}$
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is $a - b\alpha_{11}$
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is $a - b\alpha_{13}$
smooth?
If so, store.
Send (a, b) up.
Repeat.
Is $a - b\alpha_{14}$
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is $a - b\alpha_{15}$
smooth?
If so, store.
Send (a, b) left.
Repeat.

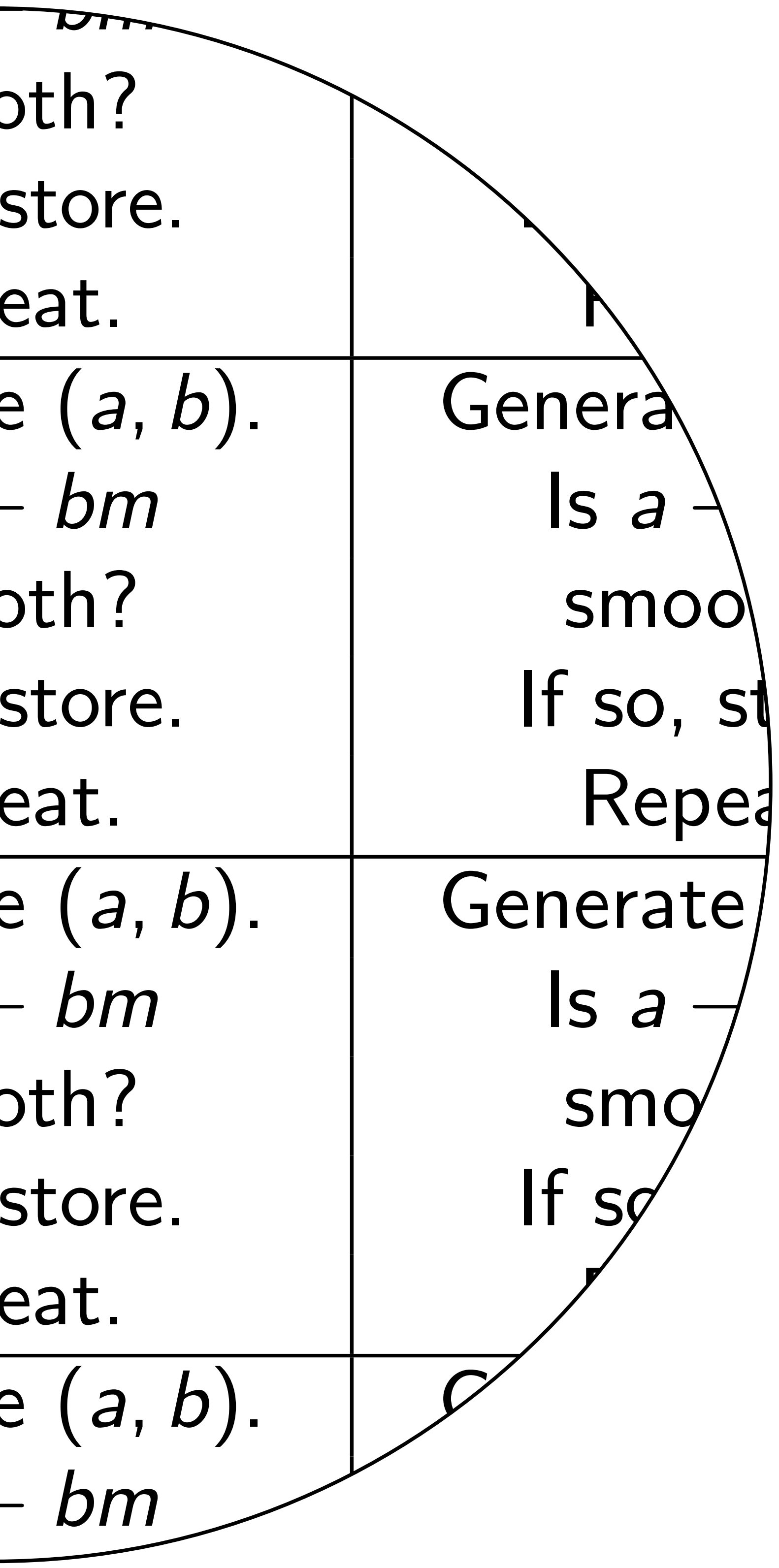


Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.

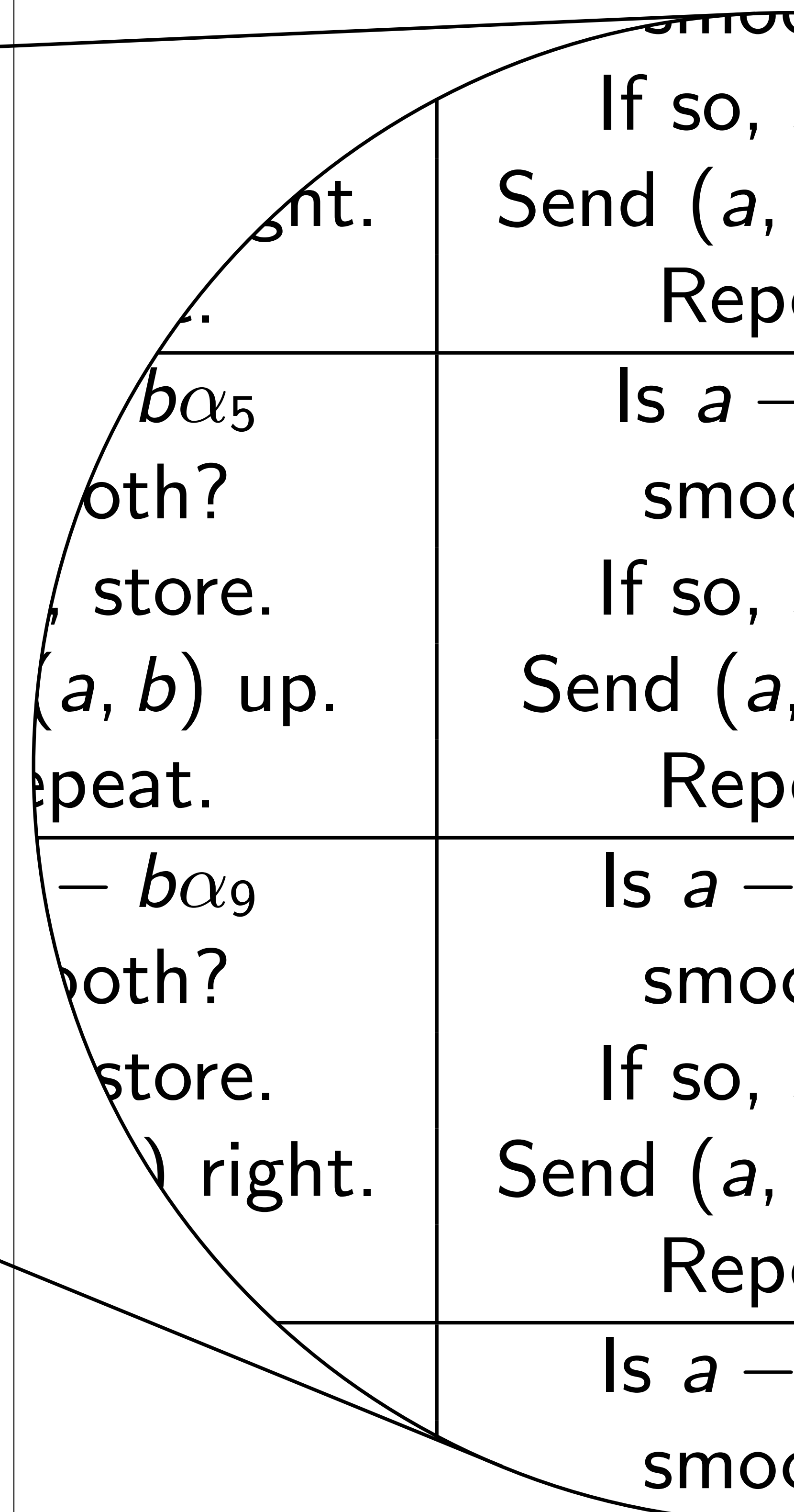


Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.





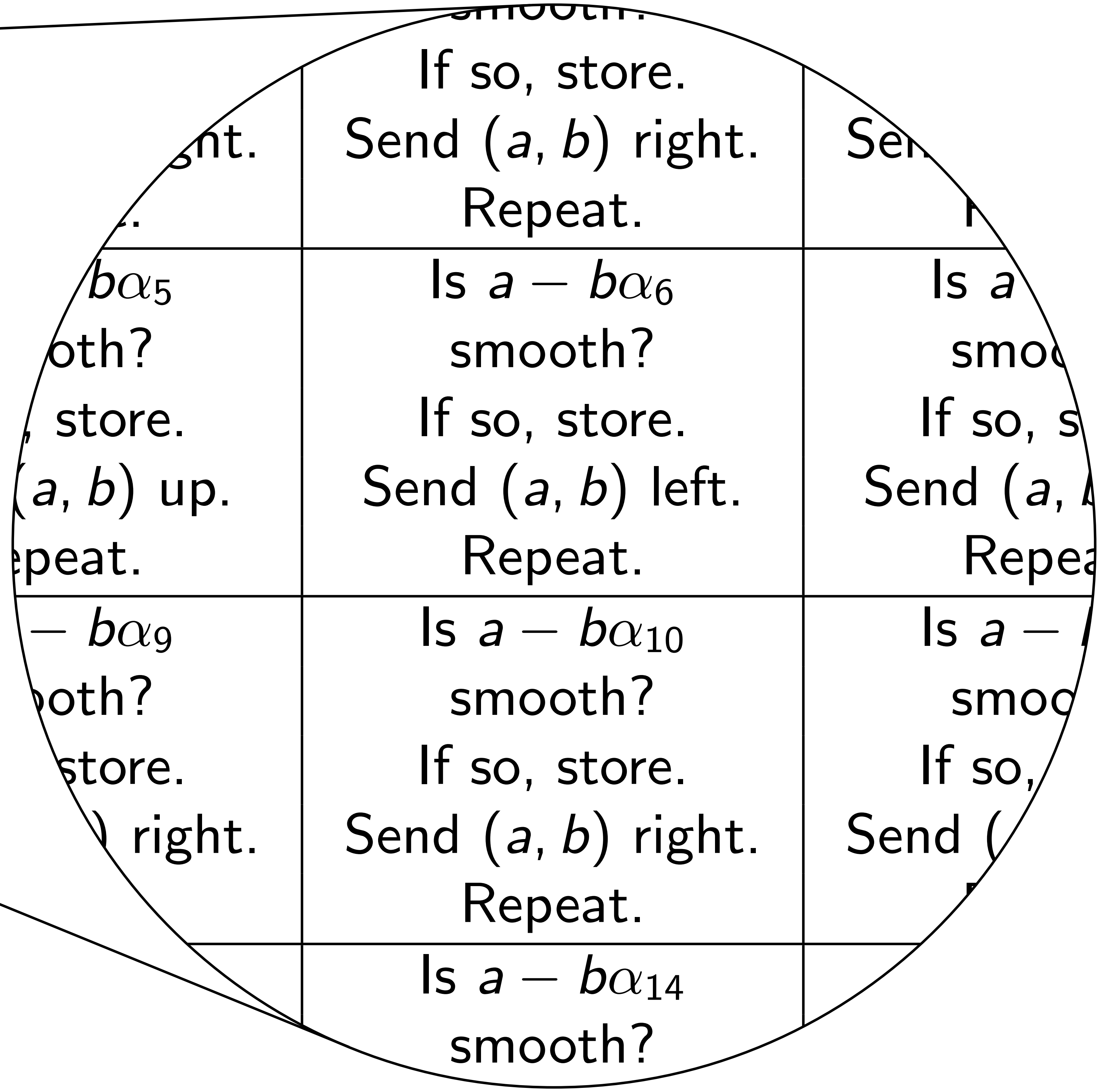
Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.

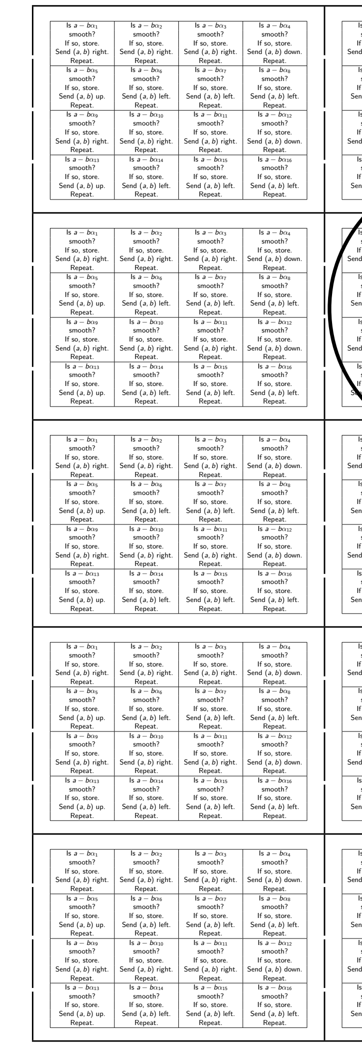


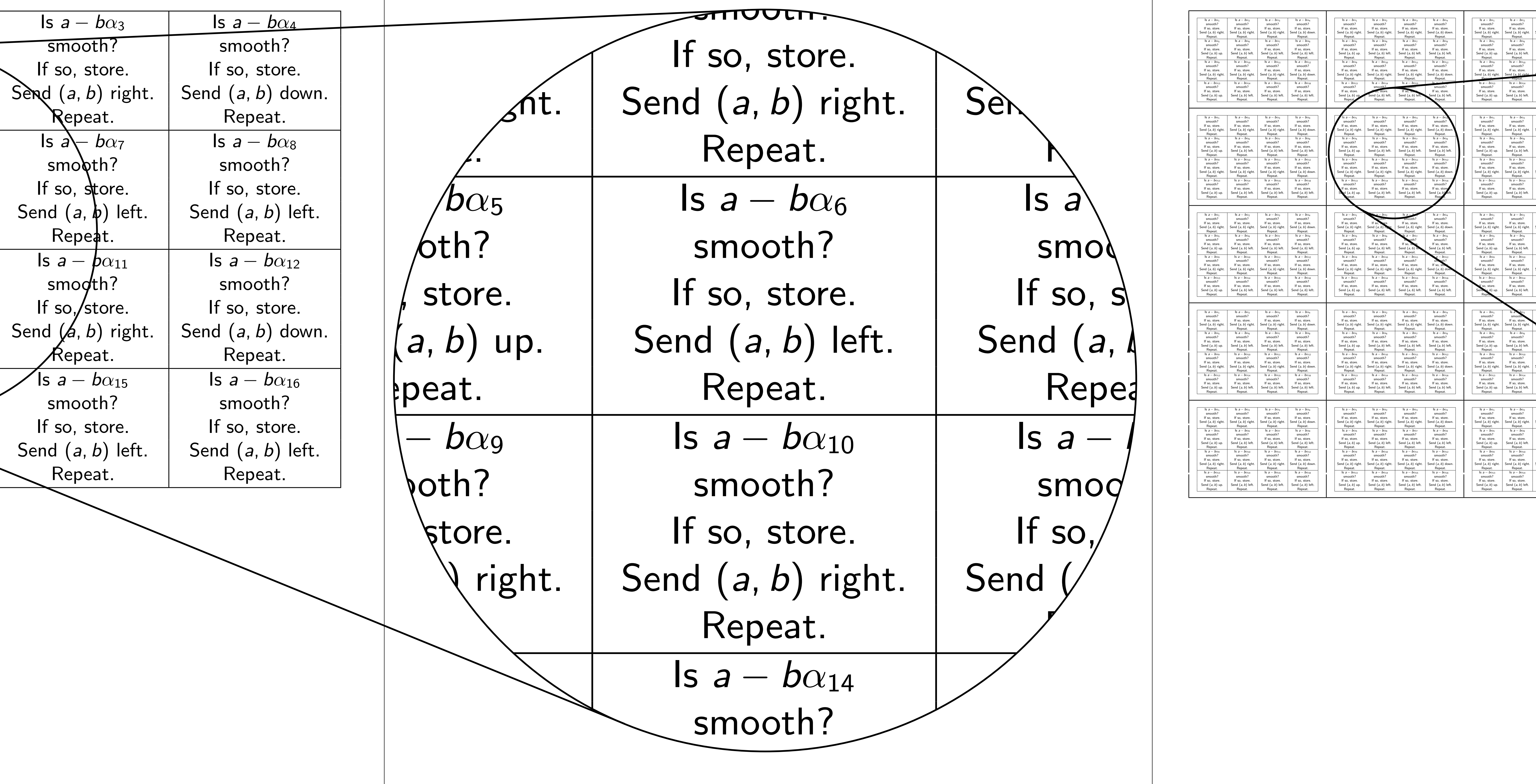
Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.

Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.

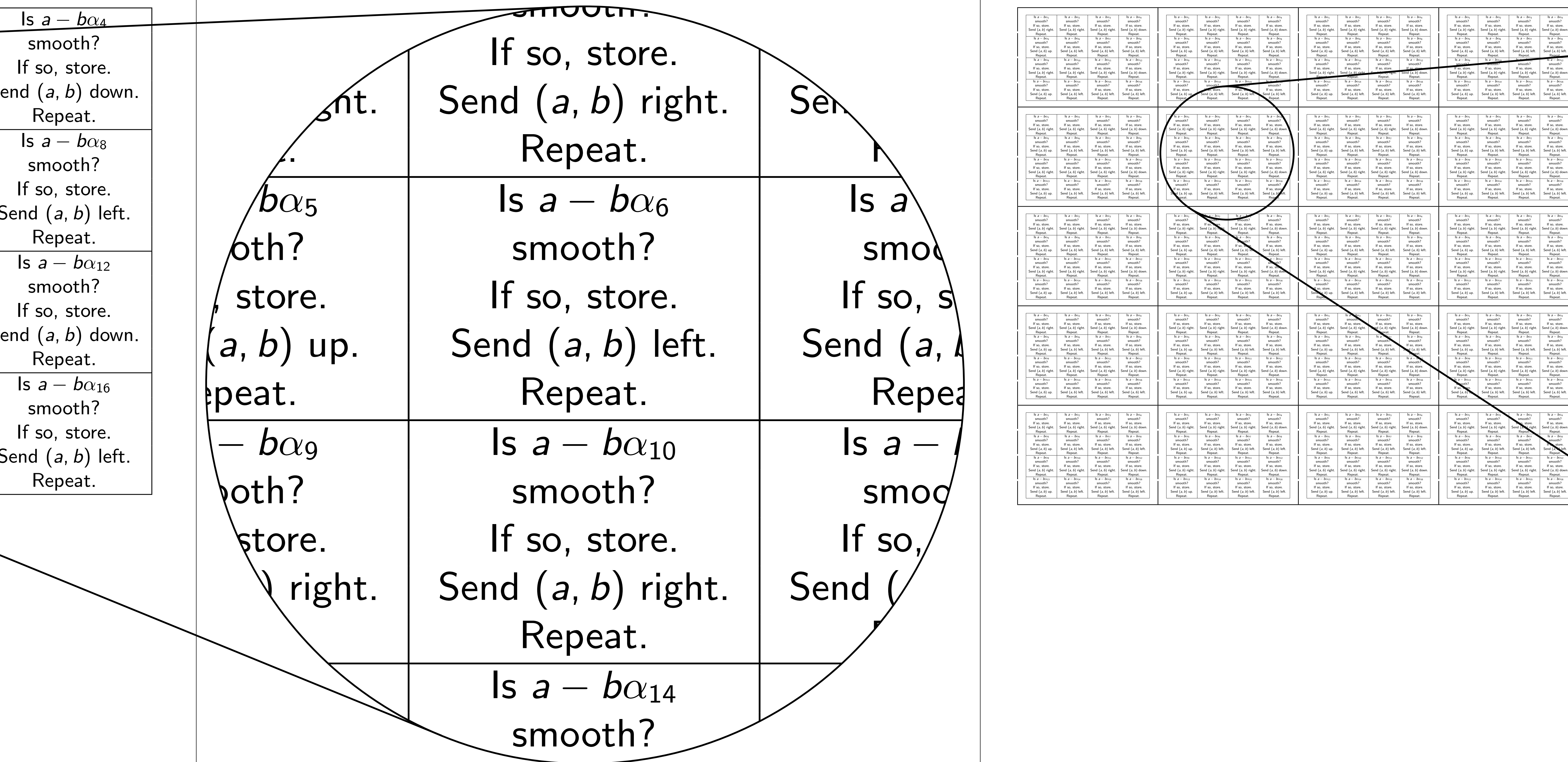
Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.



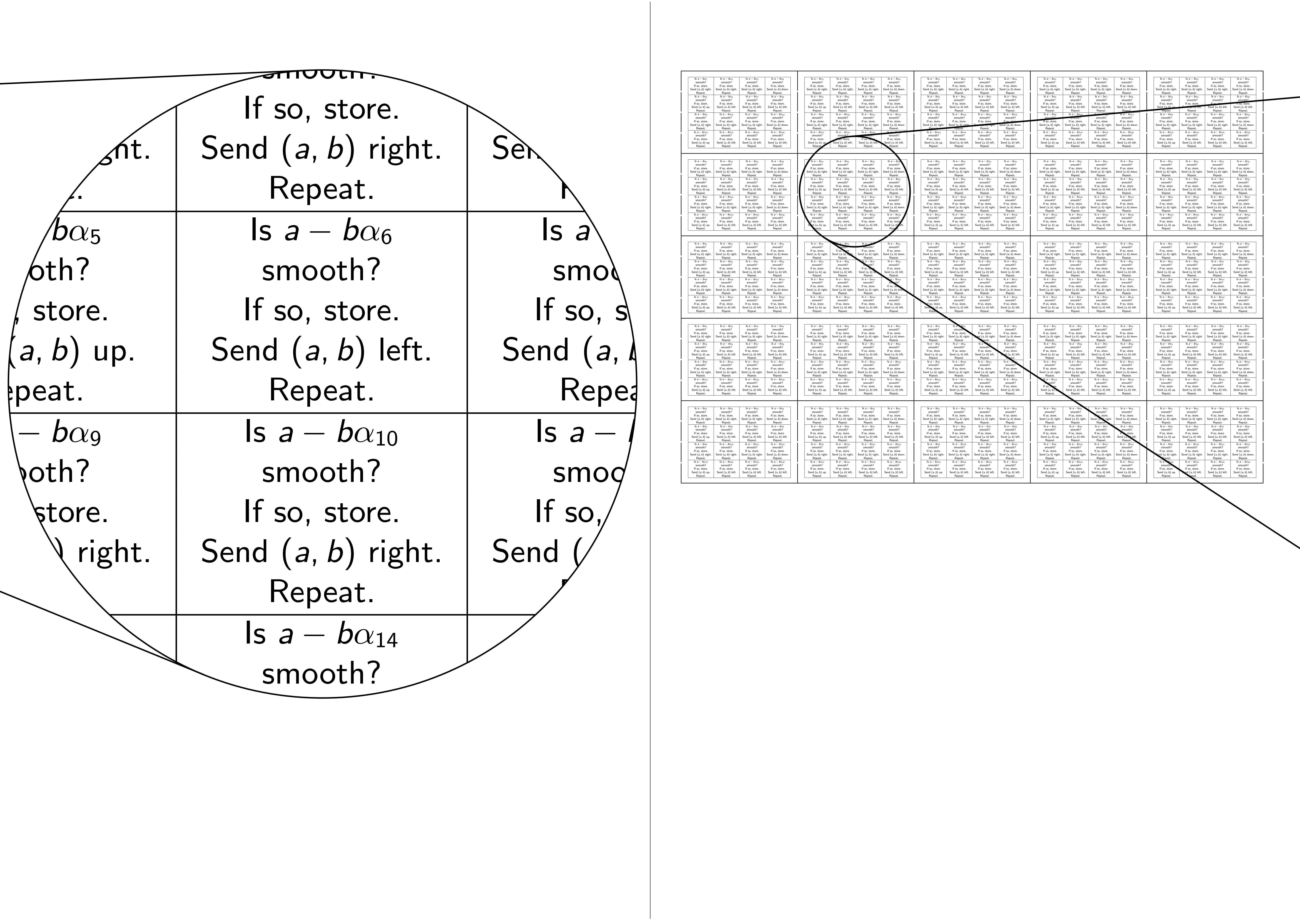


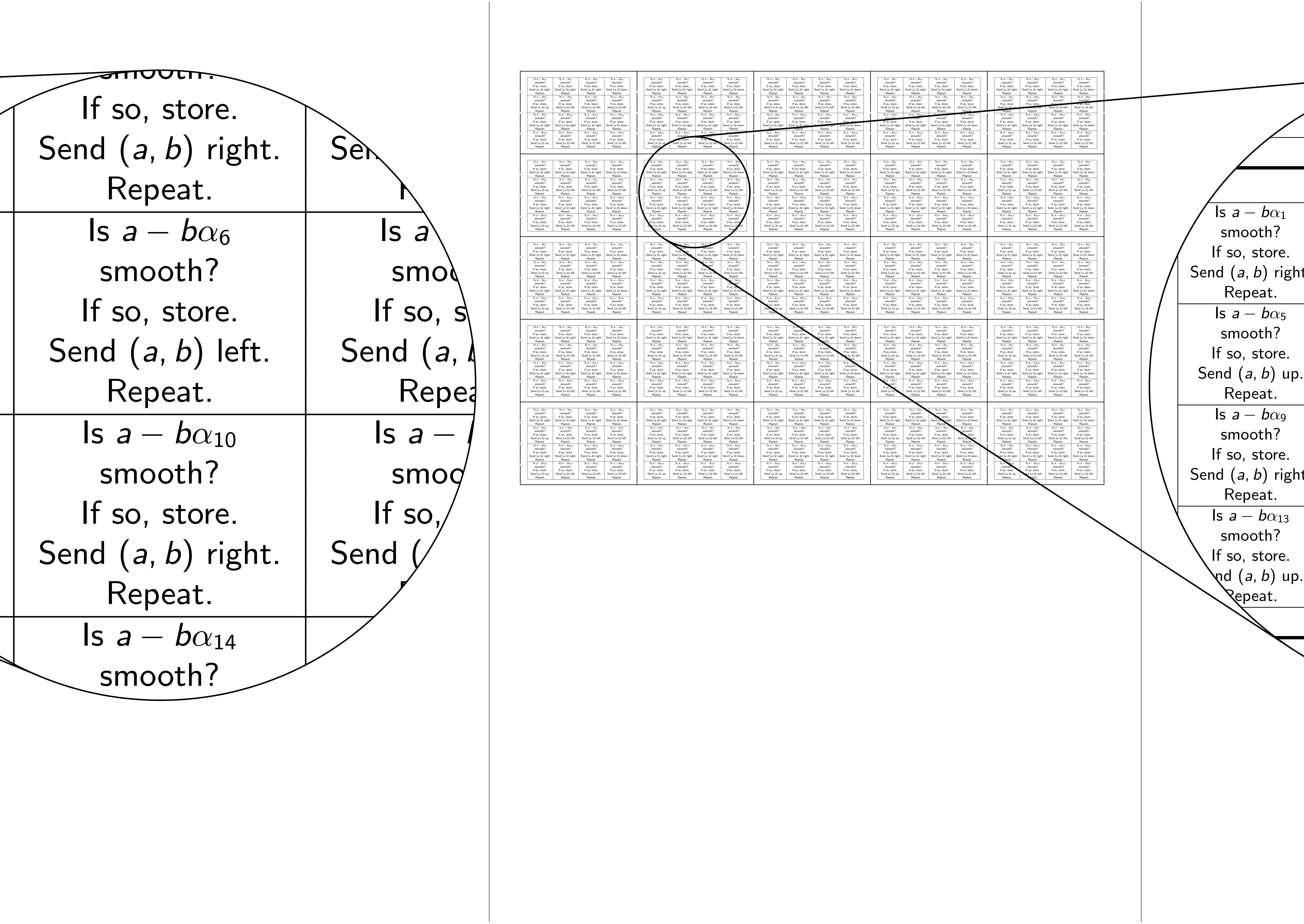


Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) left. Repeat.
If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{14}$ smooth?

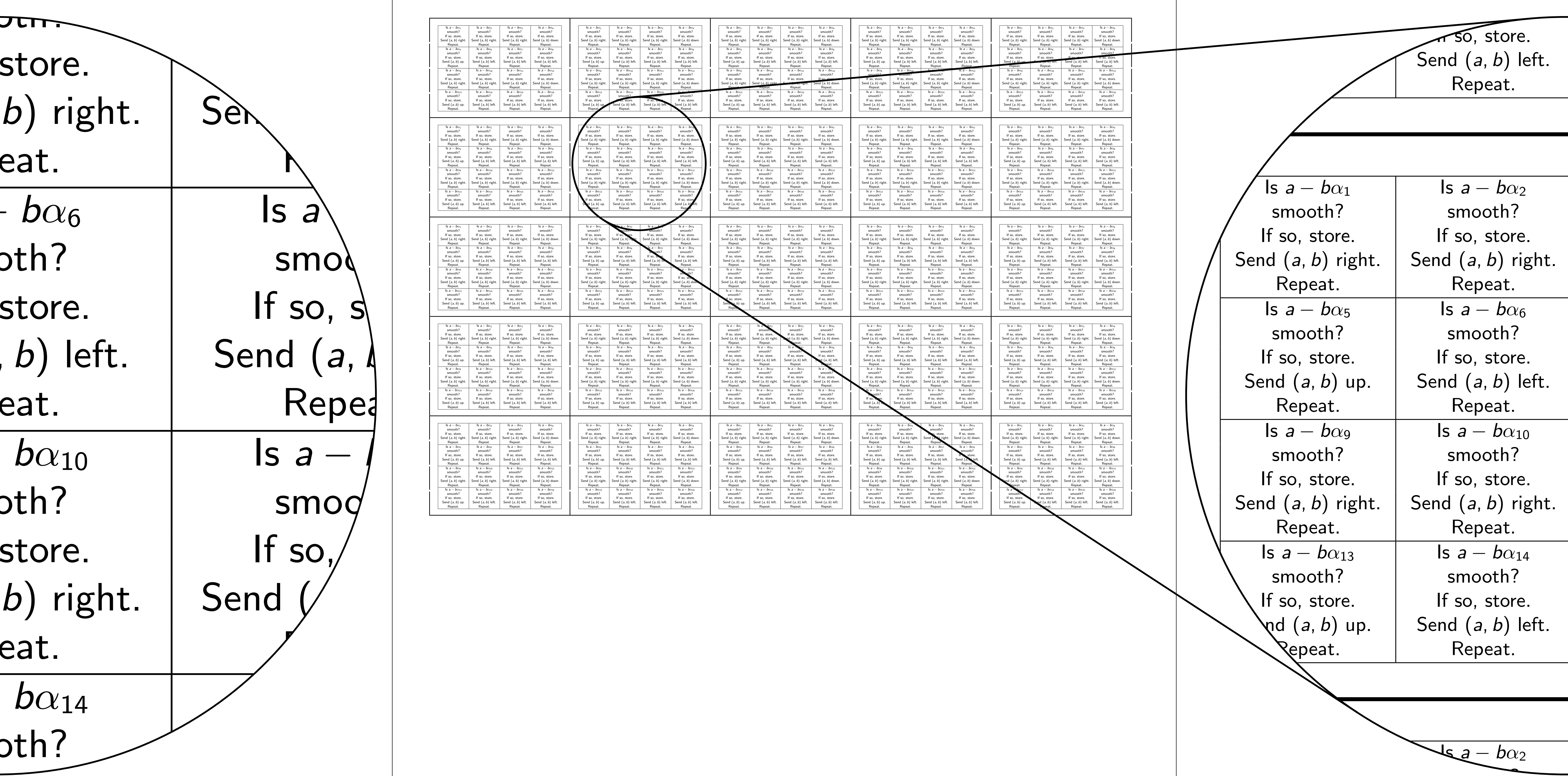


Is a − bα4
smooth?
If so, store.
end (a, b) down.
Repeat.
Is a − bα8
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα12
smooth?
If so, store.
end (a, b) down.
Repeat.
Is a − bα16
smooth?
If so, store.
Send (a, b) left.
Repeat.
If so, store.
Send (a, b) right.
Repeat.
bα5
, store.
(a, b) up.
Is a − bα6
smooth?
If so, store.
Send (a, b) left.
Repeat.
− bα9
store.
) right.
Is a − bα10
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα14
smooth?

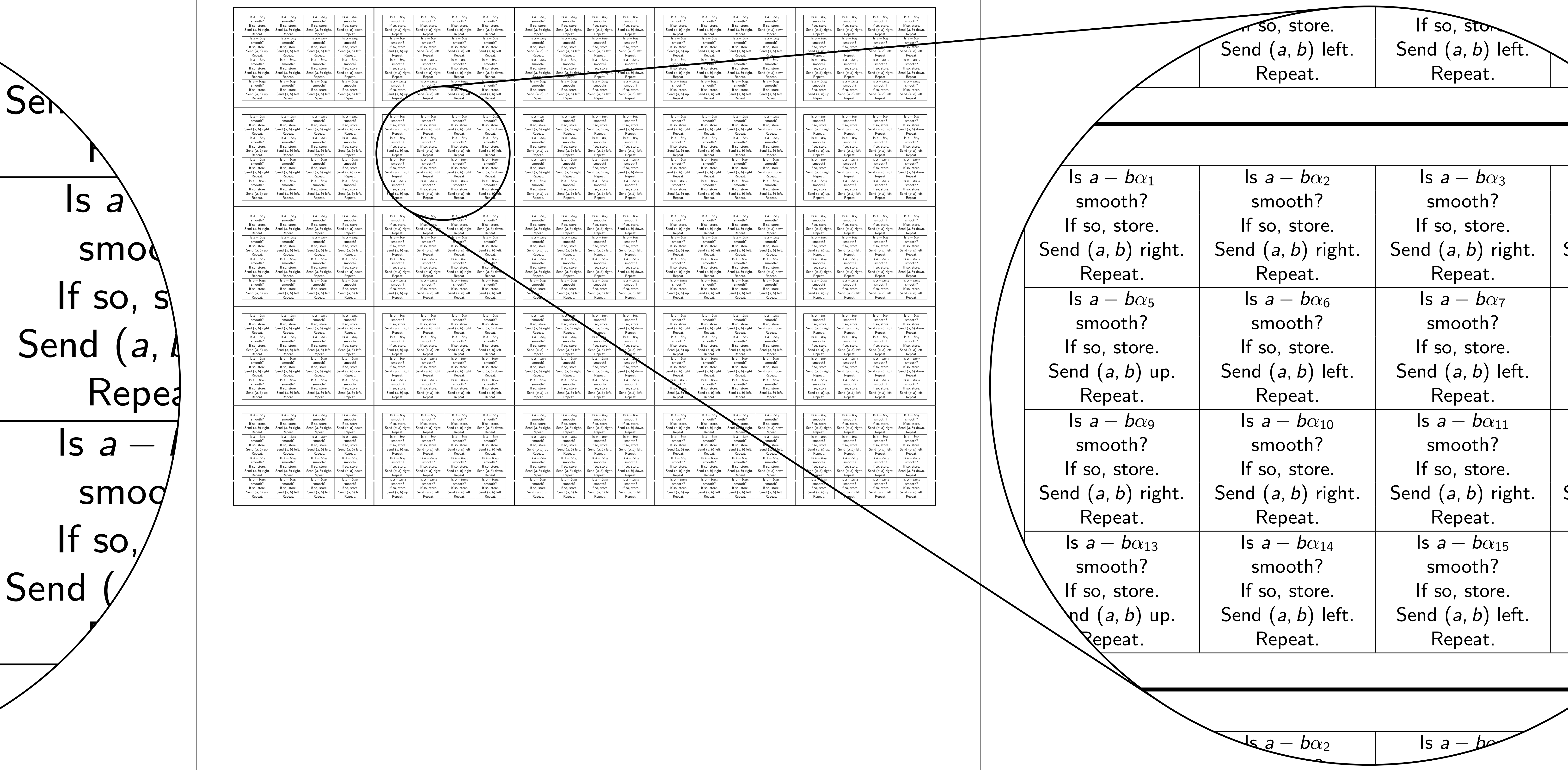

If so, store.
Send (a, b) right.
Repeat.
Is a − bα6
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα10
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα14
smooth?



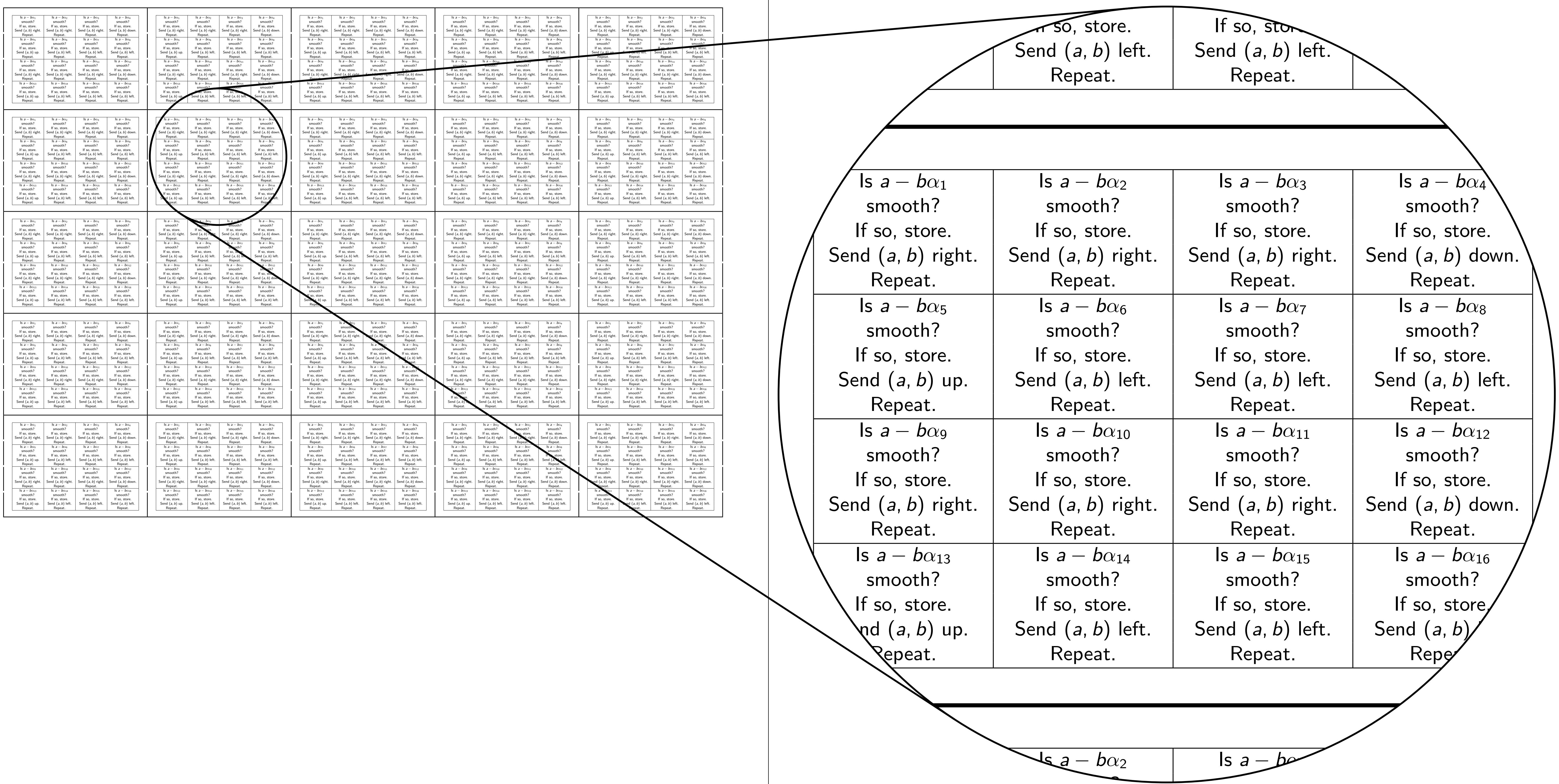
If so, store.
Send (a, b) right.
Repeat.
Is a − bα6
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα10
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα14
smooth?
Is a − bα1
smooth?
If so, store.
Repeat.
Is a − bα5
smooth?
If so, store.
Send (a, b) up.
Repeat.
Is a − bα9
smooth?
If so, store.
Repeat.
Is a − bα13
smooth?
If so, store.



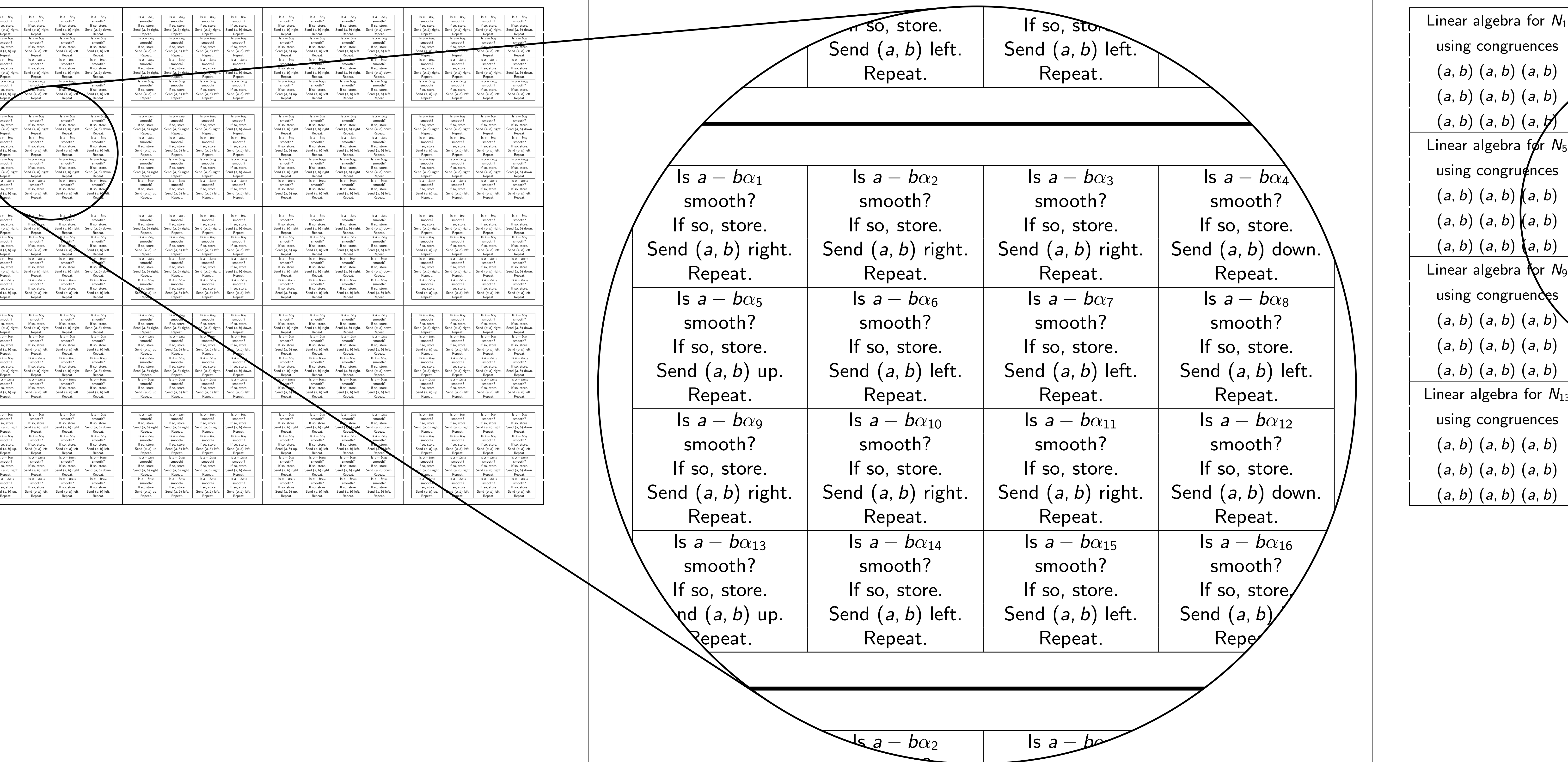
Send (a, b) left.
Repeat.
Is a − bα1
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα2
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα5
smooth?
If so, store.
Send (a, b) up.
Repeat.
Is a − bα6
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα9
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα10
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα13
smooth?
If so, store.
Is a − bα14
smooth?
If so, store.
Send (a, b) left.
Repeat.



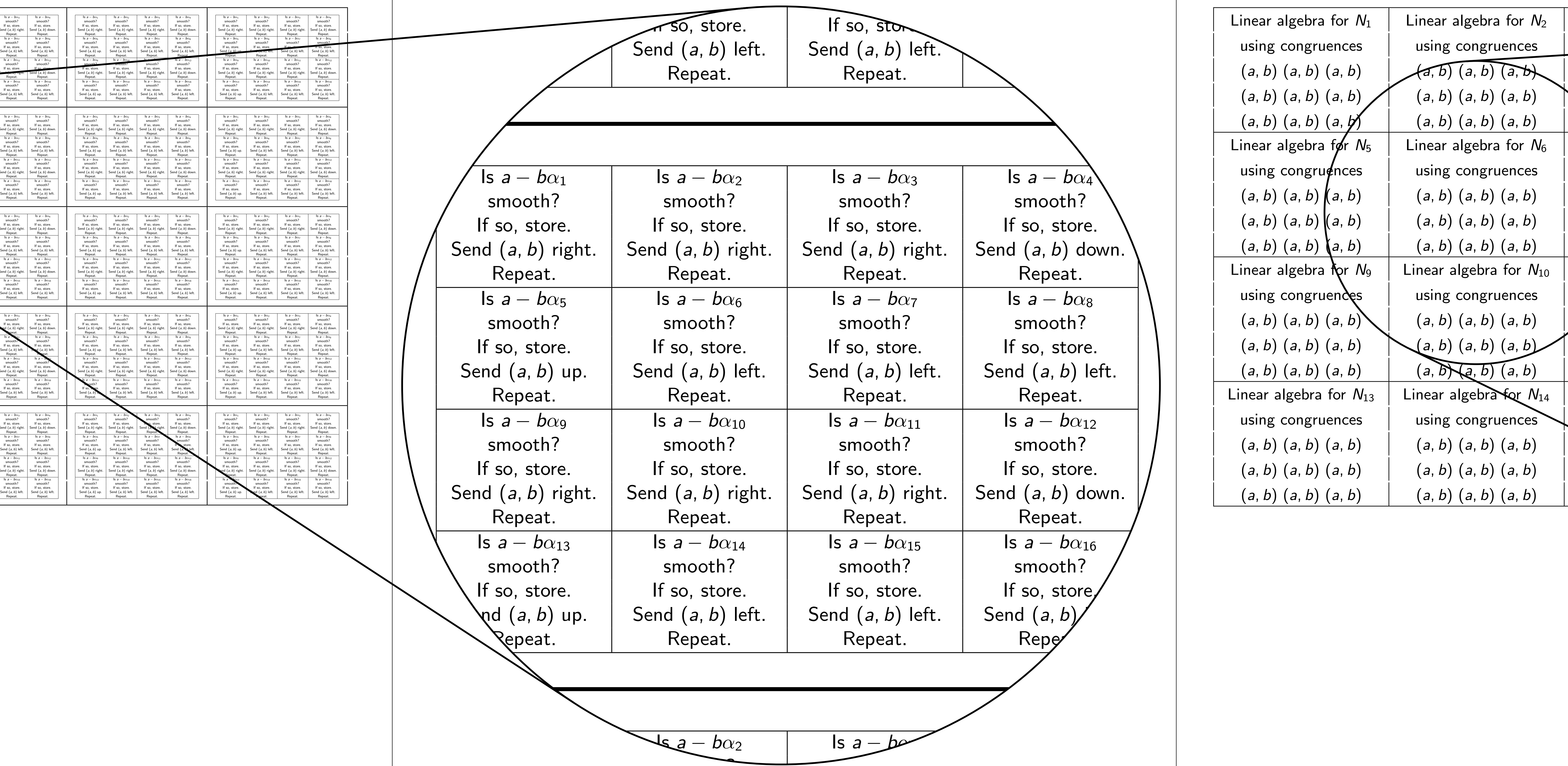
Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.
Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.
Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.



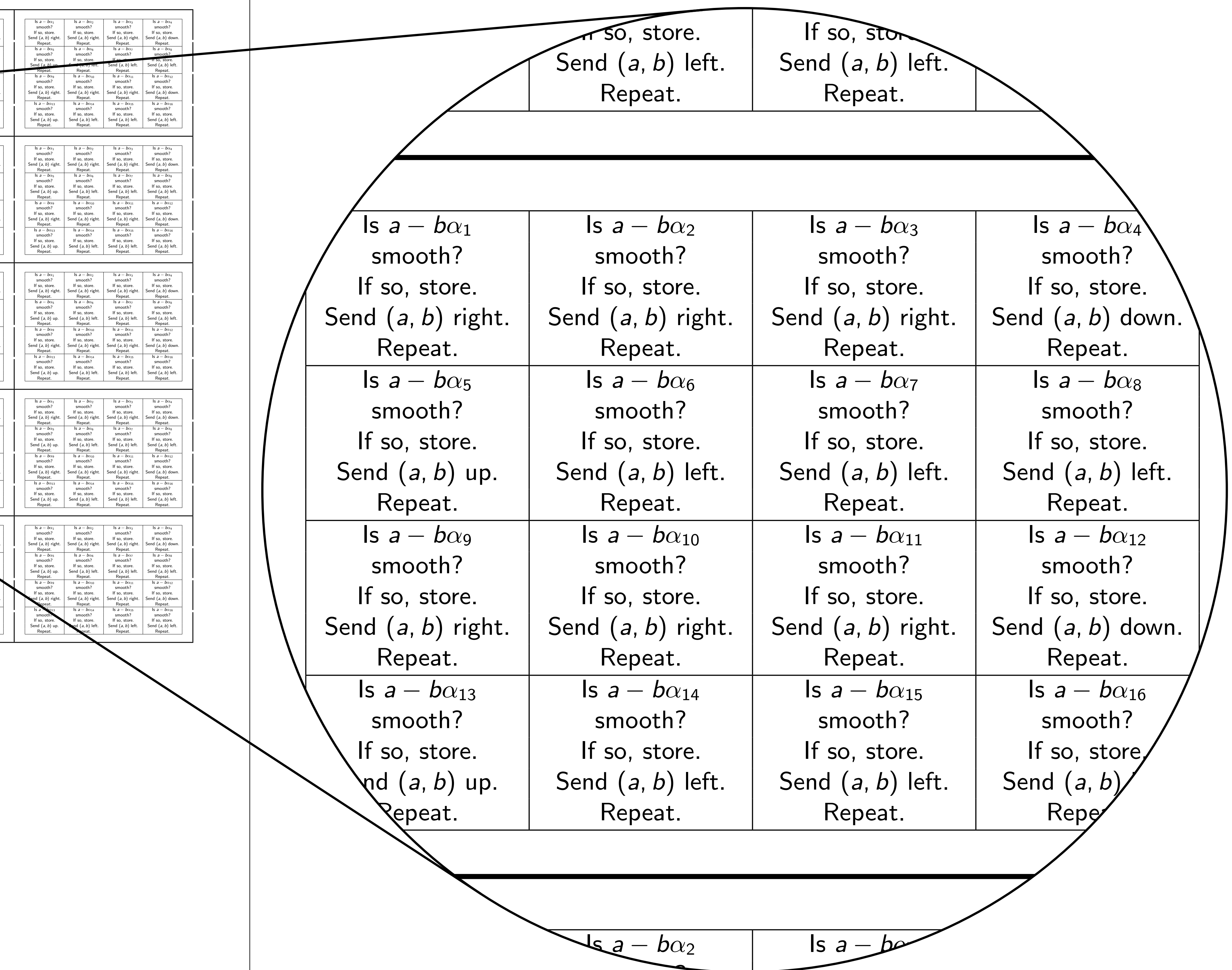
Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.
Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.
Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.
Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_{16}$ smooth? If so, store.



If so, store.
Send (a, b) left.
Repeat.
Is a − bα1
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα2
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα3
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα4
smooth?
If so, store.
Send (a, b) down.
Repeat.
Is a − bα5
smooth?
If so, store.
Send (a, b) up.
Repeat.
Is a − bα6
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα7
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα8
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα9
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα10
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα11
smooth?
If so, store.
Send (a, b) right.
Repeat.
Is a − bα12
smooth?
If so, store.
Send (a, b) down.
Repeat.
Is a − bα13
smooth?
If so, store.
Is a − bα14
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα15
smooth?
If so, store.
Send (a, b) left.
Repeat.
Is a − bα16
smooth?
If so, store.
Is a − bα2
Linear algebra for N1
using congruences
(a, b) (a, b) (a, b)
Linear algebra for N5
using congruences
Linear algebra for N9
using congruences
using congruences



Send (a, b) left.
Repeat.
If so, store.
Send (a, b) left.
Repeat.
Is a − bα1 smooth? If so, store. Send (a, b) right. Repeat.
Is a − bα2 smooth? If so, store. Send (a, b) right. Repeat.
Is a − bα3 smooth? If so, store. Send (a, b) right. Repeat.
Is a − bα4 smooth? If so, store. Send (a, b) down. Repeat.
Is a − bα5 smooth? If so, store. Send (a, b) up. Repeat.
Is a − bα6 smooth? If so, store. Send (a, b) left. Repeat.
Is a − bα7 smooth? If so, store. Send (a, b) left. Repeat.
Is a − bα8 smooth? If so, store. Send (a, b) left. Repeat.
Is a − bα9 smooth? If so, store. Send (a, b) right. Repeat.
Is a − bα10 smooth? If so, store. Send (a, b) right. Repeat.
Is a − bα11 smooth? If so, store. Send (a, b) right. Repeat.
Is a − bα12 smooth? If so, store. Send (a, b) down. Repeat.
Is a − bα13 smooth? If so, store.
Is a − bα14 smooth? If so, store. Send (a, b) left. Repeat.
Is a − bα15 smooth? If so, store. Send (a, b) left. Repeat.
Is a − bα16 smooth?
Is a − bα2
Linear algebra for N1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N10 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N13 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N14 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)



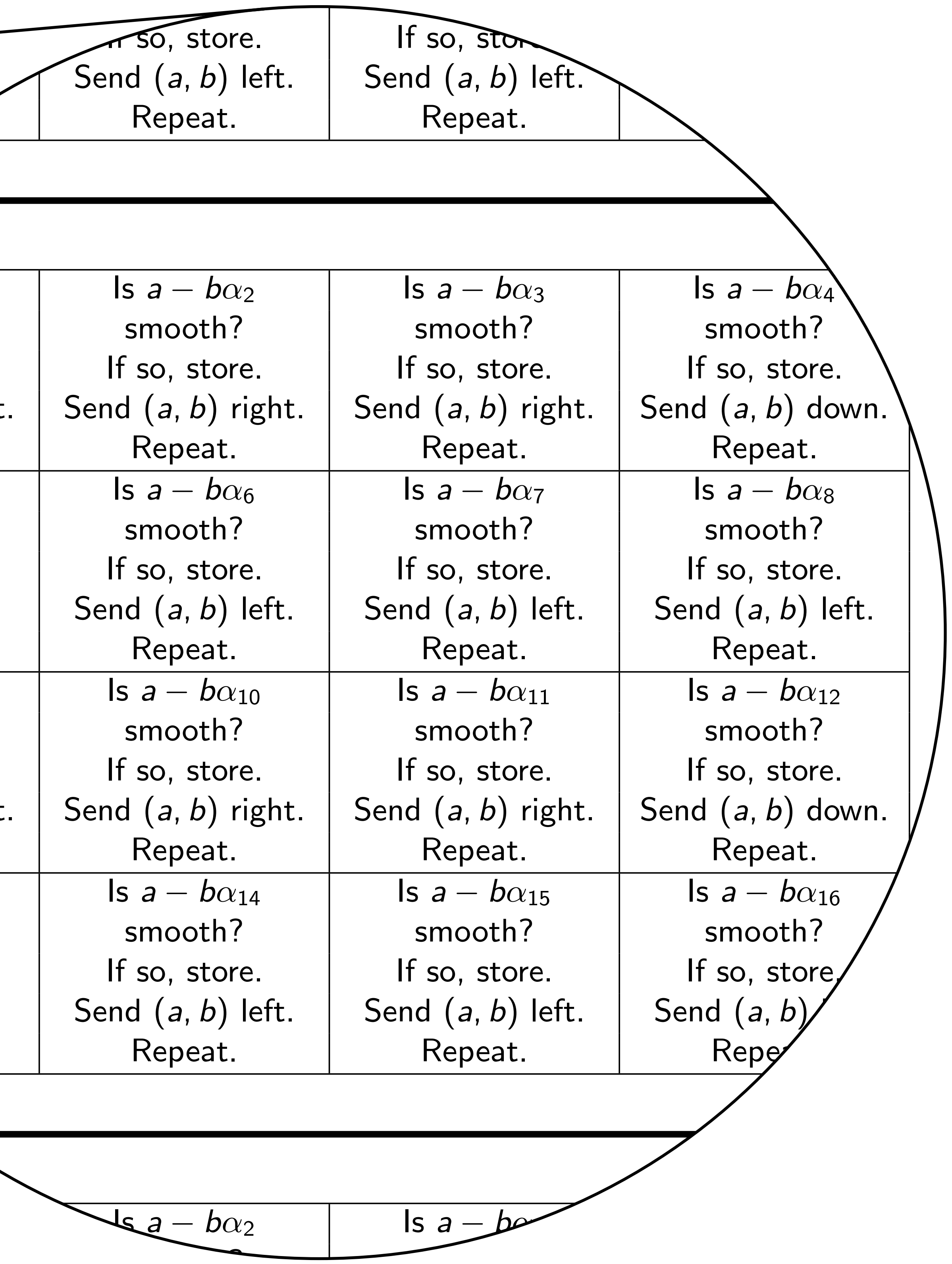
Linear algebra for N_1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_3 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	L
Linear algebra for N_5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_7 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	L
Linear algebra for N_9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{10} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{11} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Li
Linear algebra for N_{13} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{14} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{15} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Li

… so, store. Send (a, b) left. Repeat. | If so, store… Send (a, b) left. Repeat.

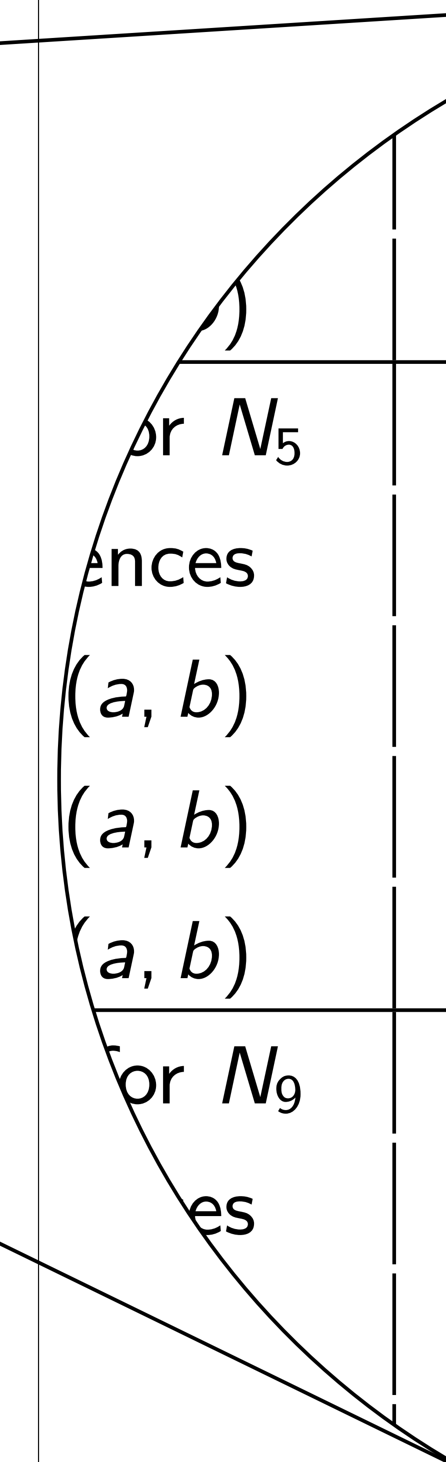
Is $a - b\alpha_1$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a, b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a, b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a, b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a, b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a, b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a, b) … Repe…

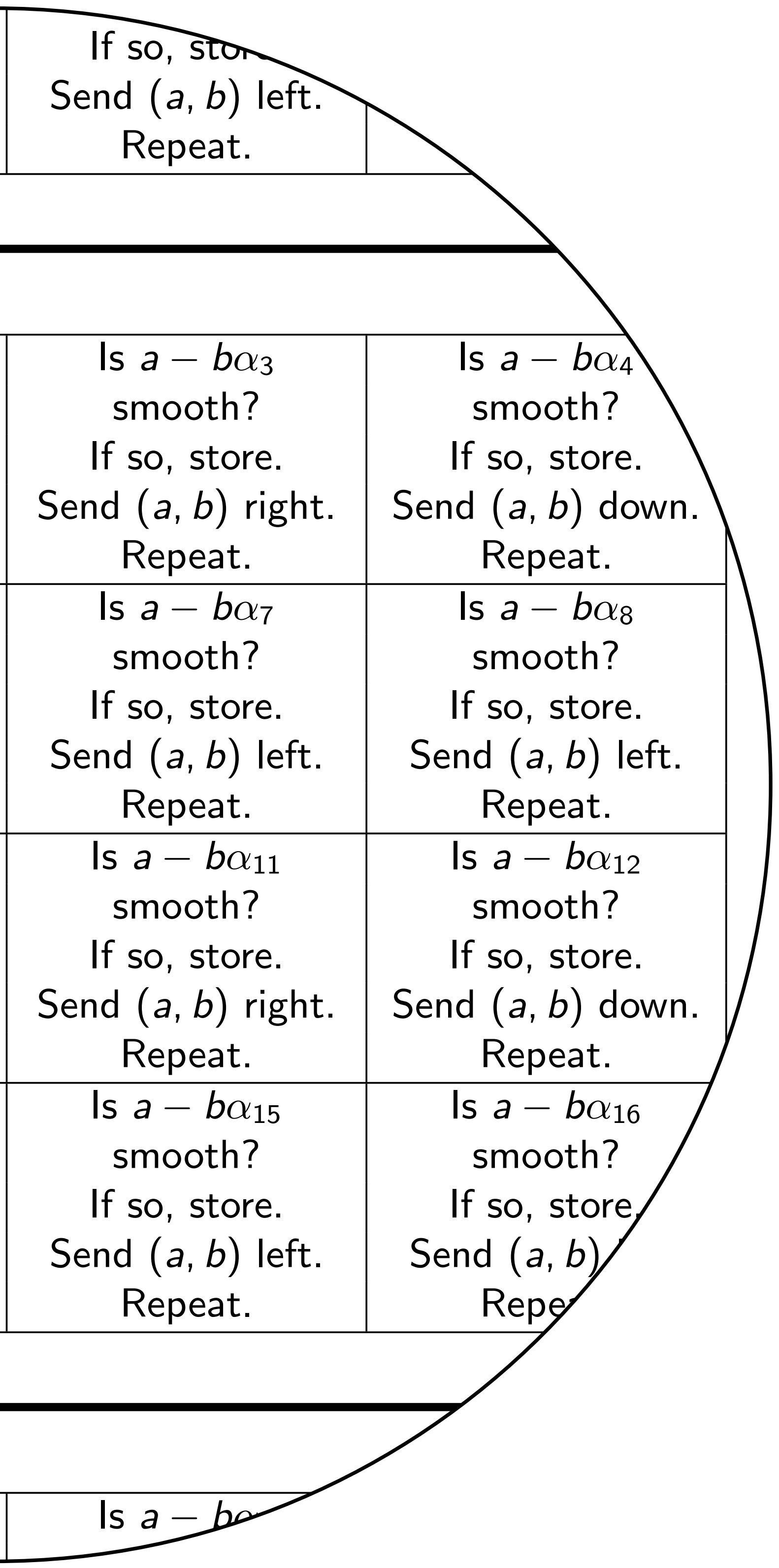
Is $a - b\alpha_2$ | Is $a - b$…

Linear algebra for N_1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_3 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_4 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_7 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_8 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{10} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{11} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{12} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_{13} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{14} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{15} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{16} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)

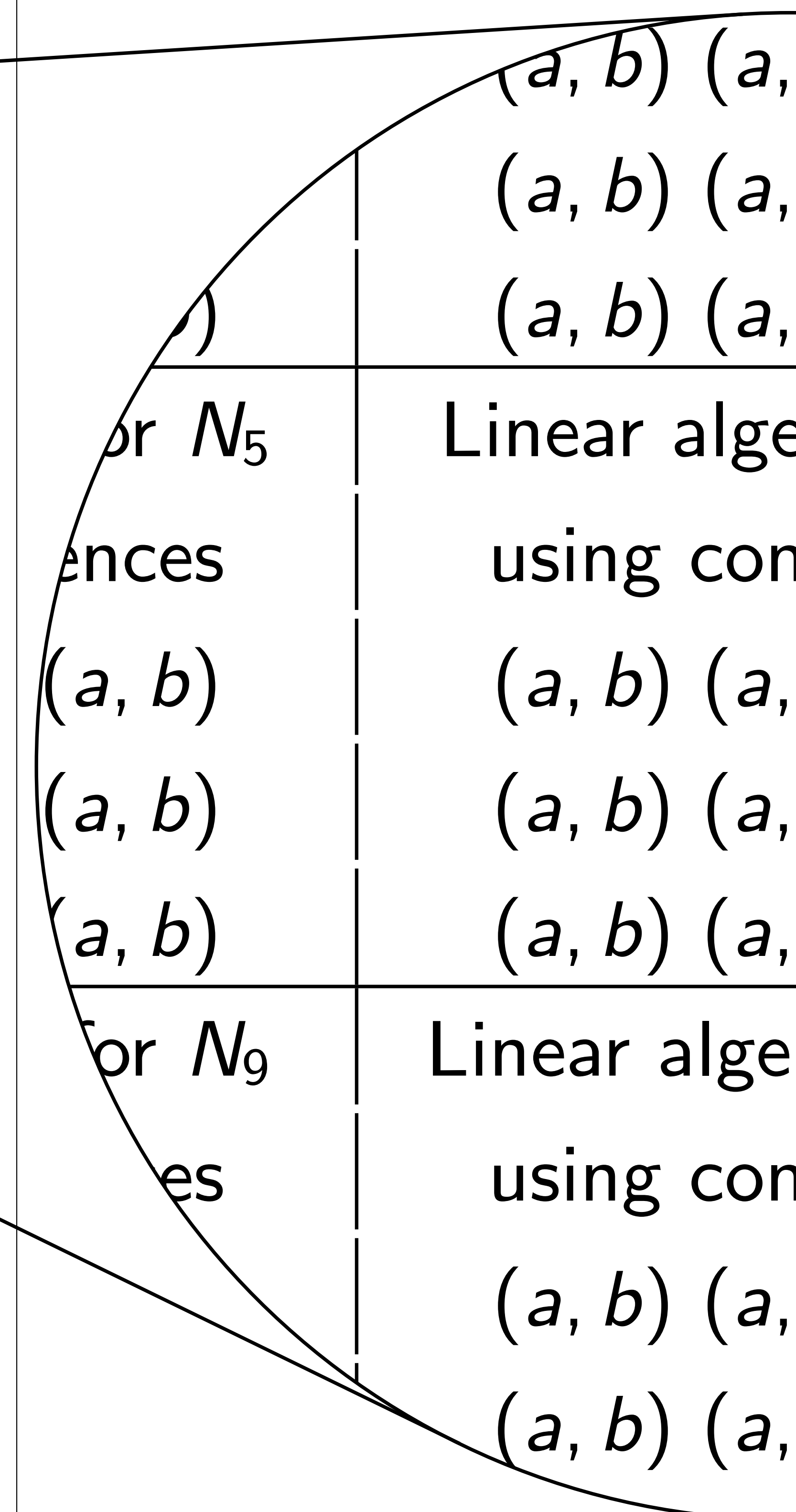


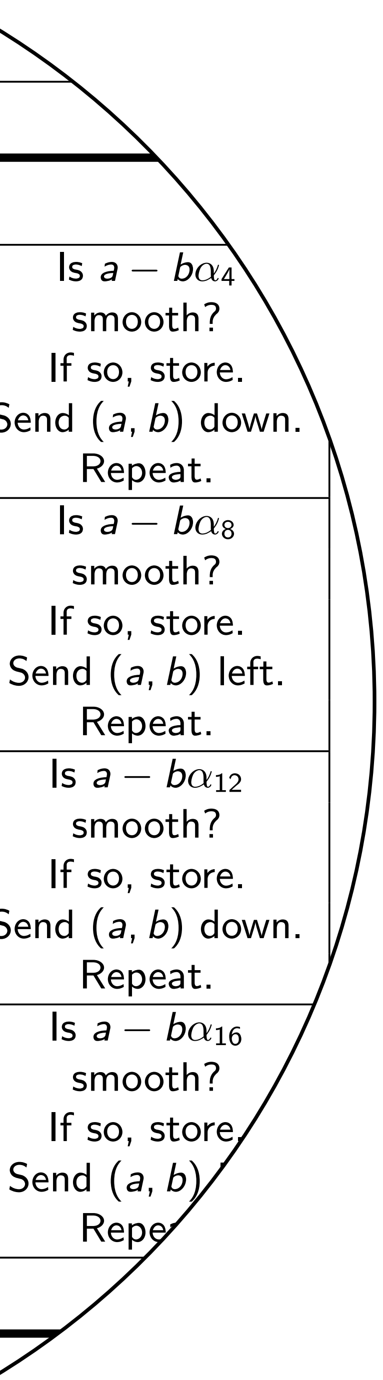
Linear algebra for N_1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_3 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_4 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_7 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_8 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{10} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{11} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{12} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_{13} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{14} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{15} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{16} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)



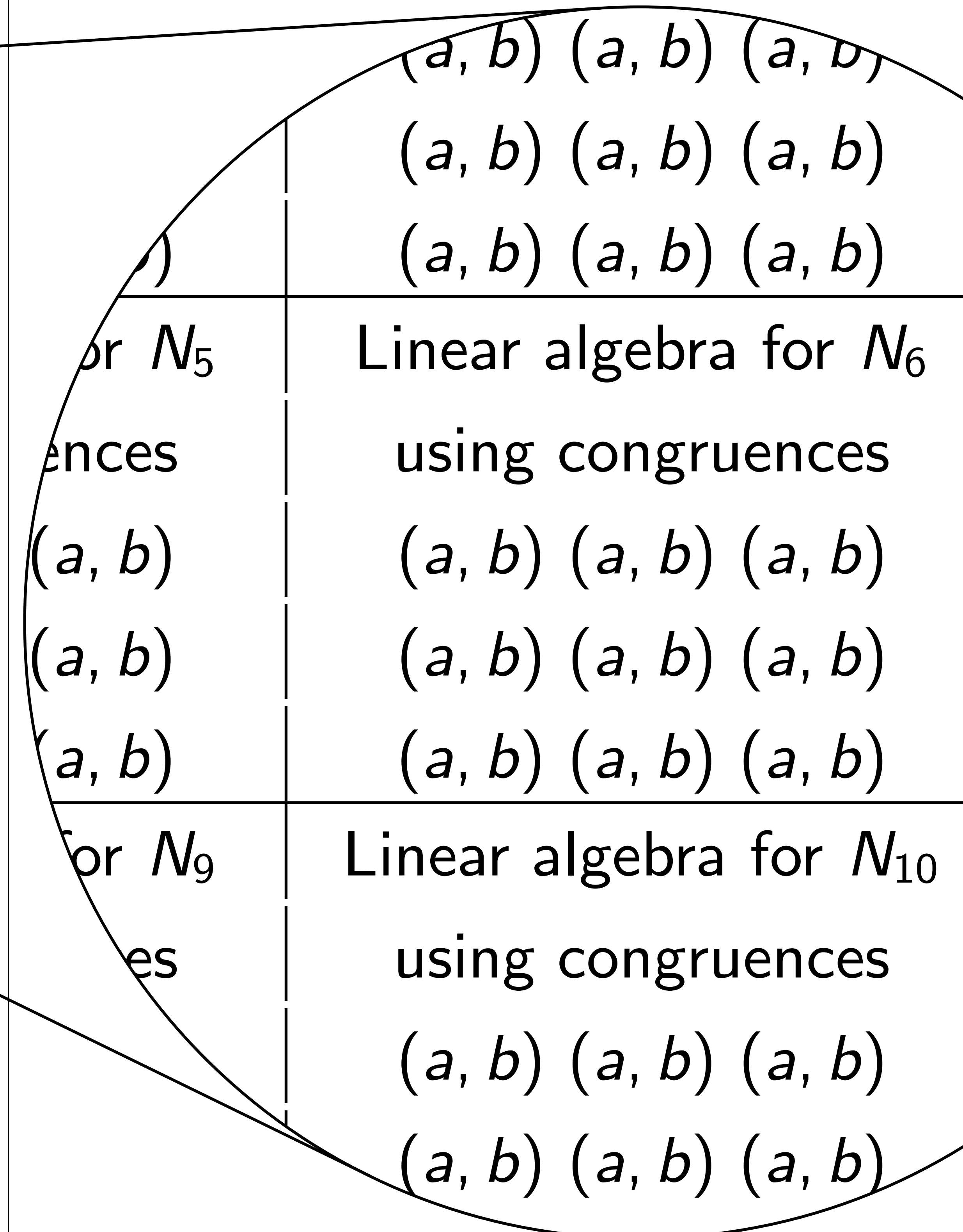


Linear algebra for N_1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_3 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_4 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_7 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_8 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{10} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{11} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{12} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_{13} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{14} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{15} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{16} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)





Linear algebra for N_1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_3 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_4 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_7 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_8 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{10} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{11} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{12} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_{13} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{14} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{15} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{16} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)



Linear algebra for N_1 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_2 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_3 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_4 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_5 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_6 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_7 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_8 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_9 using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{10} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{11} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{12} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)
Linear algebra for N_{13} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{14} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{15} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)	Linear algebra for N_{16} using congruences (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b) (a, b)