Conditional SNeRE Policies
SNeRG Technical Note 39

Stuart C. Shapiro
Department of Computer Science and Engineering
and Center for Cognitive Science
201 Bell Hall
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
`shapiro@cse.buffalo.edu`

December 15, 2005

1 Introduction

This is an investigation into the techniques of writing conditional `wheneverdo` and `whendo` policies in SNeRE. For each policy, we will state the correct behaviors for a conditional policy, show SNePSLOG interactions testing each technique on each behavior, and point out which technique performed the behavior correctly. Finally, for each policy, we will state succinctly the proper technique for implementing conditional policies.

In each test run

- the following prologue will be used, but not shown

```
set-mode-3
untrace inference acting
define-frame say (action line)
define-frame InState (nil instate)
define-frame P (nil p)
^^
(define-primaction say (line)
  (format t "~&~A~%" (sneps:choose.ns line)))
(attach-primaction believe believe disbelieve disbelieve say say)
^^
```

- the SNePSLOG line echoing each assertion will not be shown.
- `list-asserted-wffs` will be done before the triggering proposition is `add`ed, so it can be seen what propositions are believed and what policies are adopted.

2 An Investigation into Conditional `wheneverdo` Policies

We will investigate two techniques for conditional `wheneverdo` policies, namely for some state *s*, proposition *p*, and act *a*:

1. `s => wheneverdo(p, a)`
2. `wheneverdo(s, believe(wheneverdo(p, a)))`

We will refer to these as the implication technique, and the nested policy technique, respectively.
The behaviors we want from a conditional `wheneverdo` policy are:

1. When *s* doesn't hold, but *p* is `add`ed, *a* shouldn't be done.
2. When *s* does hold, and *p* is `add`ed, *a* should be done.
3. After *a* is done once, as long as *s* continues to hold, when *p* is `add`ed, *a* should be done.
4. When *s* ceases to hold, and then *p* is added, *a* should not be done.
5. When *s* ceases to hold, but is then `add`ed again, and then *p* is added, *a* should be done.

We will test each technique on each of the behaviors.

1. When *s* doesn't hold, but *p* is `add`ed, *a* shouldn't be done.

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))

: P(1)!
  P(1)

:
```

   Both techniques performed this behavior correctly.

2. When *s* does hold, and *p* is `add`ed, *a* should be done.

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  wheneverdo(P(1),say(Act triggered by =>))
  InState(s1)

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

:
```

   Again, both techniques performed this behavior correctly.

3. After `a` is done once, as long as `s` continues to hold, when `p` is `add`ed, `a` should be done.

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  wheneverdo(P(1),say(Act triggered by =>))
  P(1)
  InState(s1)

: P(1)!
  P(1)

:
```

This did not work because the message-passing processes that implement SNePS inference will not send the exact same message, that `p` has been `add`ed, twice.

One fix is to do a `clear-infer` before the second `add` of *p*:

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

: clear-infer
Node activation cleared. Some register information retained.

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  wheneverdo(P(1),say(Act triggered by =>))
  P(1)
  InState(s1)

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

:
```

A second way to get the correct behavior is to do a `disbelieve(`*p*`)` before subsequent `add`s.

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

: perform disbelieve(P(1))

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  wheneverdo(P(1),say(Act triggered by =>))
  InState(s1)

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

:
```

A third way to get the correct behavior is to add `p` for the second and subsequent times by performing `believe(p)`:

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: P(1)!
Act triggered by =>
Act triggered by wheneverdo

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  wheneverdo(P(1),say(Act triggered by =>))
  P(1)
  InState(s1)

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

:
```

Both the implication technique and the nested policy technique perform this behavior correctly as long as either: a `clear-infer` is done before second and subsequent `adds` of `p`; `disbelieve(p)` is performed before second and subsequent `adds` of `p`; or `p` is added for the second and subsequent times by performing `believe(p)`. In the remaining tests, we will always add `p` by performing `believe(p)`.

4. When *s* ceases to hold, and then *p* is added, *a* should not be done.

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: perform disbelieve(InState(s1))

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  P(1)

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

:
```

The fact that the implication technique behaves incorrectly is a mystery, and is probably a bug in SNePS. Note that its policy was not currently adopted.

The fact that the nested policy technique behaves incorrectly is understandable. Note that its policy is still adopted. The policy was adopted when the state was entered, and remained adopted when the state was `disbelieved`. This is the desired behavior for `wheneverdo(`*s*`, believe(`*p*`))`.

What we need is a companion, cancelling policy when the state s definitely doesn't holds:

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).
: wheneverdo(~InState(s1),
             disbelieve(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: InState(s1)!
  wheneverdo(P(1),say(Act triggered by wheneverdo))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: perform believe(~InState(s1))

: list-asserted-wffs
  wheneverdo(~InState(s1),
             disbelieve(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  ~InState(s1)
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  P(1)

: perform believe(P(1))
Act triggered by =>

:
```

Now the implication technique still, mysteriously, performed incorrectly, but the nested policy technique did perform correctly. In all subsequent tests, we will add and remove s by performing `believe(`s`)` and `believe(`$\sim s$`)`.

5. When *s* ceases to hold, but is then added again, and then *p* is added, *a* should be done.

```
: InState(s1) => wheneverdo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(wheneverdo(P(1), say("Act triggered by wheneverdo")))).
: wheneverdo(~InState(s1),
             disbelieve(wheneverdo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: perform believe(~InState(s1))

: perform believe(P(1))
Act triggered by =>

: perform believe(InState(s1))
Act triggered by wheneverdo

: list-asserted-wffs
  wheneverdo(~InState(s1),
             disbelieve(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(InState(s1),
             believe(wheneverdo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => wheneverdo(P(1),say(Act triggered by =>))
  wheneverdo(P(1),say(Act triggered by =>))
  P(1)
  InState(s1)

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

:
```

Both techniques performed this behavior correctly.

3 Conclusion: Conditional `wheneverdo` Policies

If you want to say that while state *s* holds, the agent should have the policy that whenever proposition *p*, is `add`ed, it should perform act *a*, you should express this as

```
wheneverdo(s, believe(wheneverdo(p,a)))
wheneverdo(~s, disbelieve(wheneverdo(p,a)))
```

You should enter and leave the state *s* by performing

`believe(s)`
and `believe(~s)`

You should add *p* by either:

- always performing `believe(p)`;
- performing `clear-infer` before the second and subsequent adds of *p*;
- or performing `disbelieve(p)` before the second and subsequent adds of *p*.

4 An Investigation into Conditional `whendo` Policies

We will investigate two techniques for conditional `whendo` policies, namely for some state *s*, proposition *p*, and act *a*:

1. `s => whendo(p, a)`
2. `wheneverdo(s, believe(whendo(p, a)))`

The behaviors we want from a conditional `whendo` policy are:

1. When *s* doesn't hold, but *p* is `add`ed, *a* shouldn't be done.
2. When *s* does hold, and *p* is `add`ed, *a* should be done.
3. After *a* is done once, even though *s* continues to hold, when *p* is `add`ed, *a* should not be done.
4. When *s* ceases to hold, and then *p* is added, *a* should not be done.
5. When *s* ceases to hold, but is then `add`ed again, and then *p* is added, *a* should be done.

We will test the two techniques on each of the behaviors.

1. When *s* doesn't hold, but *p* is `add`ed, *a* shouldn't be done.

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  InState(s1) => whendo(P(1),say(Act triggered by =>))

: perform believe(P(1))

:
```

So, both techniques perform this behavior correctly.

2. When *s* does hold, and *p* is `add`ed, *a* should be done.

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  whendo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => whendo(P(1),say(Act triggered by =>))
  whendo(P(1),say(Act triggered by =>))
  InState(s1)

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

:
```

So, again, both techniques perform this behavior correctly.

3. After *a* is done once, even though *s* continues to hold, when *p* is `add`ed, *a* should not be done.

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  InState(s1) => whendo(P(1),say(Act triggered by =>))
  whendo(P(1),say(Act triggered by =>))
  P(1)
  InState(s1)

: perform believe(P(1))
Act triggered by =>

:
```

Here, the implication technique still, and incorrectly, performed the act for a second consecutive time. Note that its policy is still adopted. It cannot be unadopted because it is considered a derived belief whose support set is still believed. This is an indication that the implication technique is semantically incorrect. We do not want the adoption of the policy to logically depend on the state, but want the policy to be adopted when the state starts to hold, and be unadopted when it is first triggered.

The nested policy technique performed correctly.

4. When *s* ceases to hold, and then *p* is added, *a* should not be done.

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: perform disbelieve(InState(s1))

: list-asserted-wffs
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  whendo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => whendo(P(1),say(Act triggered by =>))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo
:
```

The fact that the implication technique behaves incorrectly is a mystery, and is probably a bug in SNePS.

The fact that the nested policy technique behaves incorrectly is understandable. Note that its policy is still adopted. The policy was adopted when the state was entered, and remained adopted when the state was `disbelieve`d. This is the desired behavior for `wheneverdo(`*s*`, believe(`*p*`))`.

What we need is a companion, cancelling policy when the state *s* definitely doesn't holds:

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).
: wheneverdo(~InState(s1),
             disbelieve(whendo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: perform believe(~InState(s1))

: list-asserted-wffs
  wheneverdo(~InState(s1),
             disbelieve(whendo(P(1),say(Act triggered by wheneverdo))))
  ~InState(s1)
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  InState(s1) => whendo(P(1),say(Act triggered by =>))

: perform believe(P(1))
Act triggered by =>

:
```

The implication technique still mysteriously doesn't behave correctly, but the nested policy technique now does behave correctly.

5. When *s* ceases to hold, but is then added again, and then *p* is added, *a* should be done.

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).
: wheneverdo(~InState(s1),
             disbelieve(whendo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: perform believe(~InState(s1))

: perform believe(P(1))
Act triggered by =>

: list-asserted-wffs
  wheneverdo(~InState(s1),
             disbelieve(whendo(P(1),say(Act triggered by wheneverdo))))
  ~InState(s1)
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  InState(s1) => whendo(P(1),say(Act triggered by =>))
  P(1)

: perform believe(InState(s1))
Act triggered by wheneverdo

: perform believe(P(1))
Act triggered by =>

:
```

Again the implication technique mysteriously behaves incorrectly.

This time the nested policy technique leads to *a* being done as soon as the *s* is reentered because *p* is believed at that time.

What we wanted was to `add` *p* after *s* is reentered. We'll do this by performing a `disbelieve` on *p* before reentering *s*:

```
: InState(s1) => whendo(P(1), say("Act triggered by =>")).
: wheneverdo(InState(s1),
             believe(whendo(P(1), say("Act triggered by wheneverdo")))).
: wheneverdo(~InState(s1),
             disbelieve(whendo(P(1), say("Act triggered by wheneverdo")))).

: perform believe(InState(s1))

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: perform believe(~InState(s1))

: perform believe(P(1))
Act triggered by =>

: perform disbelieve(P(1))

: perform believe(InState(s1))

: list-asserted-wffs
  wheneverdo(~InState(s1),
             disbelieve(whendo(P(1),say(Act triggered by wheneverdo))))
  wheneverdo(InState(s1),
             believe(whendo(P(1),say(Act triggered by wheneverdo))))
  whendo(P(1),say(Act triggered by wheneverdo))
  InState(s1) => whendo(P(1),say(Act triggered by =>))
  whendo(P(1),say(Act triggered by =>))
  InState(s1)

: perform believe(P(1))
Act triggered by =>
Act triggered by wheneverdo

: perform believe(P(1))
Act triggered by =>

:
```

Again, the implication technique mysteriously behaves incorrectly when *s* does not hold, and when *s* does hold and *p* is added for the second time in a row. It does perform correctly immediately after *s* is reentered. The nested policy technique, however, behaves correctly in all cases.

5 Conclusion: Conditional `whendo` Policies

If you want to say that while state *s* holds, the agent should have the policy that when proposition *p*, is next `add`ed, it should perform act *a*, you should express this as

```
wheneverdo(s, believe(whendo(p,a)))
wheneverdo(~s, disbelieve(whendo(p,a)))
```

You should enter and leave the state *s* by performing

`believe(s)`
and `believe(~s)`

You should add *p* by either:

- always performing `believe(p)`;
- performing `clear-infer` before the second and subsequent adds of *p*;
- or performing `disbelieve(p)` before the second and subsequent adds of *p*.