## Modify BNF 7.2, 7.3, A.2.2.1:

struct_union ::= **struct** | **union** [ **soft** | **tagged** ]

## Modify BNF footnote 16:

### Current:

When a packed dimension is used with the **struct** or **union** keyword, the **packed** keyword shall also be used.

### New:

When a packed dimension is used with the **struct** keyword, the **packed** keyword shall also be used. When a packed dimension is used with the **union** keyword, the **soft** and/or **packed** keyword shall also be used.

## Modify 7.3.1:

### Current:

Packed unions shall only contain members that are of integral data types. The members of a packed, untagged union shall all be the same size (in contrast to an unpacked union or a packed, tagged union, where the members can be different sizes). Thus, a union member that was written as another member can be read back. A packed union differs from an unpacked union in that when a packed union appears as a *primary*, it shall be treated as a single vector.

A packed union can also be used as a whole with arithmetic and logical operators, and its behavior is determined by its signedness, with unsigned being the default. One or more bits of a packed union can be selected as if it were a packed array with the range `[n-1:0]`.

Only packed data types and the integer data types summarized in Table 6-8 (see 6.11) shall be legal in packed unions.

If a packed union contains a 2-state member and a 4-state member, the entire union is 4-state. There is an implicit conversion from 4-state to 2-state when reading and from 2-state to 4-state when writing the 2-state member.

For example, a union can be accessible with different access widths:

```
typedef union packed { // default unsigned
    s_atmcell acell;
    bit [423:0] bit_slice;
    bit [52:0][7:0] byte_slice;
} u_atmcell;
u_atmcell u1;
byte b; bit [3:0] nib;
b = u1.bit_slice[415:408]; // same as b = u1.byte_slice[51];
nib = u1.bit_slice [423:420]; // same as nib = u1.acell.GFC;
```

With packed unions, writing one member and reading another is independent of the byte ordering of the machine, unlike an unpacked union of unpacked structures, which are C-compatible and have members in ascending address order.

### New:

Packed unions shall only contain members that are of integral data types (see 6.11.1). Unlike unpacked and tagged unions, packed untagged unions allow a union member that was written as another member to be read back. Two forms of packed untagged unions are supported: *hard packed* and *soft packed*. When the **packed** qualifier is used without the **soft** qualifier on an untagged union, the union is *hard packed* and members of that union shall all be

the same size. When the **soft** qualifier is used on an untagged union, the union is *soft packed* and members of that union do not have to be of the same size. Since the **soft** qualifier indicates that the union is *soft packed*, the **packed** qualifier may be omitted when the **soft** qualifier is used.

~~The members of a packed, untagged union shall all be the same size. (in contrast to an unpacked union, or a packed, tagged union, where the members can be different sizes). Thus, a union member that was written as another member can be read back. A packed union differs from an unpacked union in that when a packed union appears as a *primary*, it shall be treated as a single vector.~~

A packed union differs from an unpacked union in that when a packed union appears as a *primary*, it shall be treated as a single vector. A packed union can also be used as a whole with arithmetic and logical operators, and its behavior is determined by its signedness, with unsigned being the default. One or more bits of a packed union can be selected as if it were a packed array with the range `[n-1:0]`.

~~Only packed data types and the integer data types summarized in Table 6-8 (see 6.11) shall be legal in packed unions.~~

If a packed union contains a 2-state member and a 4-state member, the entire union is 4-state. There is an implicit conversion from 4-state to 2-state when reading and from 2-state to 4-state when writing the 2-state member.
For example, a packed union can be accessible with different access widths:

```
typedef union packed { // default unsigned
    s_atmcell acell;
    bit [423:0] bit_slice;
    bit [52:0][7:0] byte_slice;
} u_atmcell;
u_atmcell u1;
byte b; bit [3:0] nib;
b = u1.bit_slice[415:408]; // same as b = u1.byte_slice[51];
nib = u1.bit_slice [423:420]; // same as nib = u1.acell.GFC;
```

With packed unions, writing one member and reading another is independent of the byte ordering of the machine, unlike an unpacked union of unpacked structures, which are C-compatible and have members in ascending address order.

The representation for a packed union with the **soft** qualifier is the following:

— The size is equal to the number of bits needed to represent the maximum of the sizes of the members.
— The bits of each member are right-justified [i.e., towards the least significant bits (LSBs)].

The representation scheme is applied recursively to any nested soft packed unions.

For example:

```
typedef union soft packed {
  struct packed {
    bit [4:0] valA, valB, valC;
  } D1;
  struct packed {
    bit[1:0] valX;
    union soft {
      bit [9:0] F1;
      bit [7:0] F2;
    } valY;
```

```
  } D2;
} Data_u;
```

The values for the Data_u type will have the layouts shown in figure 7-1

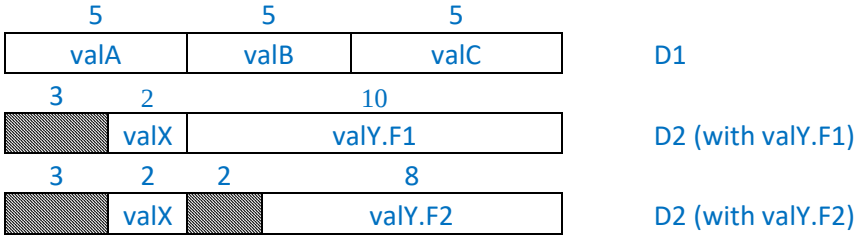


*Figure 7-1 —Data_u type with soft packed qualifier*

When assigning to a member of a packed union with the **soft** qualifier, the value of any MSBs beyond the member bits are unaffected.

*Note to the editor: The shading in the above Figure 7-1 should be carried into the pre-existing figures (now 7-2 and 7-3).*

# Modify 7.3.2 (Tagged unions):

## Current

When the **`packed`** qualifier is used on a tagged union, all the members shall have packed types, but they do not have to be of the same size. The (standard) representation for a packed tagged union is the following:
— The size is always equal to the number of bits needed to represent the tag plus the maximum of the sizes of the members.
— The size of the tag is the minimum number of bits needed to code for all the member names (e.g., five to eight members would need 3 tag bits).
— The tag bits are always left-justified (i.e., towards the MSBs).
— For each member, the member bits are always right-justified [i.e., towards the least significant bits (LSBs)].
— The bits between the tag bits and the member bits are undefined. In the extreme case of a void member, only the tag is significant and all the remaining bits are undefined.

The representation scheme is applied recursively to any nested tagged unions.

For example, if the `VInt` type definition had the **`packed`** qualifier, `Invalid` and `Valid` values will have the layouts shown in Figure 7-1, respectively.

< Figure 7-1 >

For example, if the `Instr` type had the **`packed`** qualifier, its values will have the layouts shown in Figure 7-2

< Figure 7-2 >

## New

~~When the **`packed`** qualifier is used on a tagged union, all the members shall have packed types, but they do not have to be of the same size~~

Like soft packed untagged unions, members of tagged unions with the **packed** qualifier shall have packed types, but do not have to be of the same size. The ~~(standard)~~ representation for a packed tagged union is the following:

— The size is ~~always~~ equal to the number of bits needed to represent the tag plus the maximum of the sizes of the members.
— The size of the tag is the minimum number of bits needed to code for all the member names (e.g., five to eight members would need 3 tag bits).
— The tag bits are ~~always~~ left-justified (i.e., towards the MSBs).
~~— For each member, the member bits are always right-justified [i.e., towards the least significant bits (LSBs)].~~
— The bits of each member are right-justified [i.e., towards the least significant bits (LSBs)].
— The bits between the tag bits and the member bits are undefined. In the extreme case of a void member, only the tag is significant and all the remaining bits are undefined.

The representation scheme is applied recursively to any nested tagged unions.

For example, if the `VInt` type definition had the **`packed`** qualifier, `Invalid` and `Valid` values will have the layouts shown in Figure 7-1, respectively.

< Figure 7-1 >

For example, if the `Instr` type had the **`packed`** qualifier, its values will have the layouts shown in Figure 7-2

< Figure 7-2 >